100 XP

Introduction

5 minutes

Creating a great data model is one of the most important tasks that a data analyst can perform in Microsoft Power BI. By doing this job well, you help make it easier for people to understand your data, which will make building valuable Power BI reports easier for them and for you.

The pages in this module are instructional only, no data files are provided. You have a chance to work with real data in the labs.
https://www.microsoft.com/en-us/videoplayer/embed/RWG0L2?postJsllMsg=true

A good data model offers the following benefits:

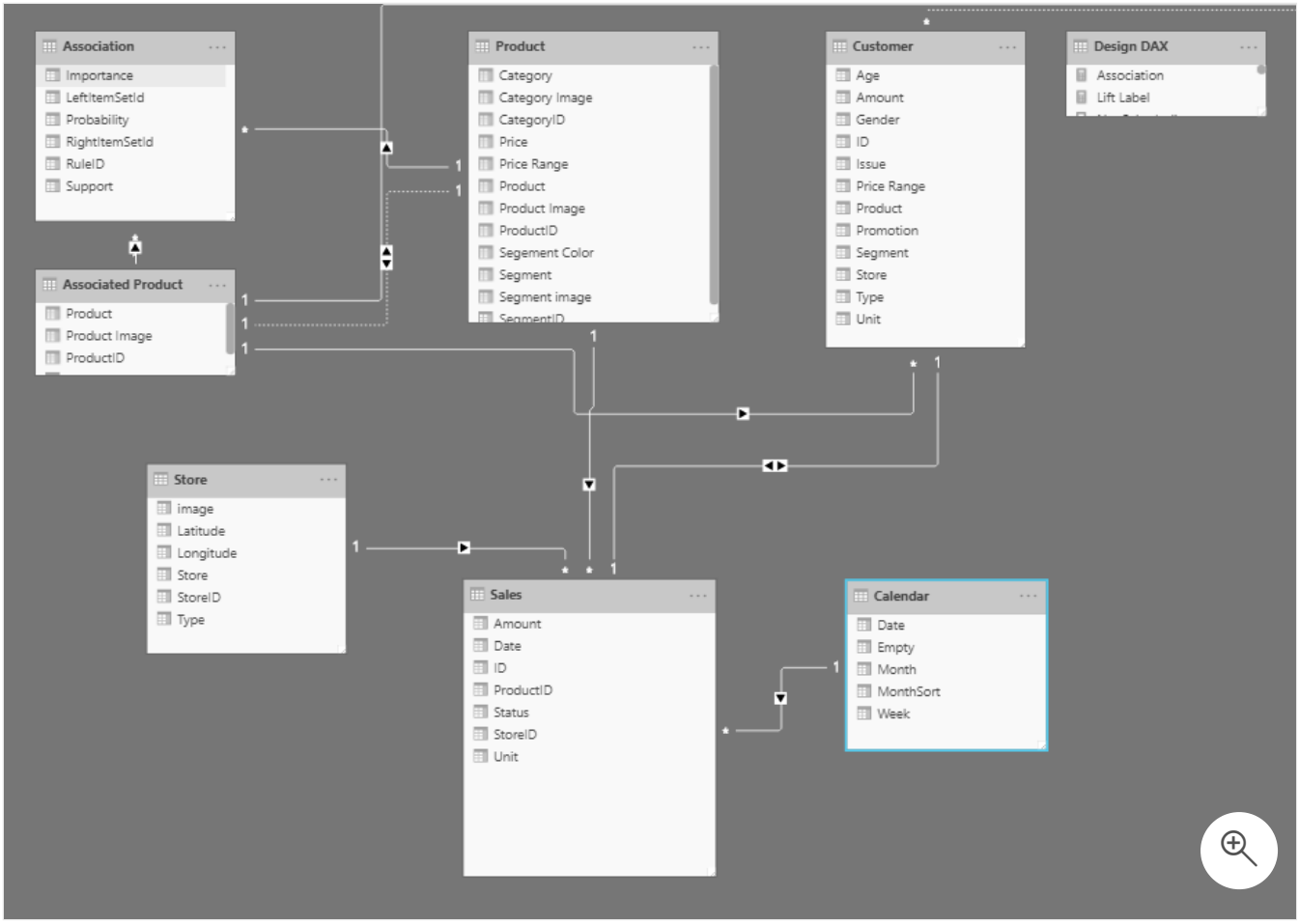
- Data exploration is faster.
- Aggregations are simpler to build.
- Reports are more accurate.
- Writing reports takes less time.
- Reports are easier to maintain in the future.

Providing set rules for what makes a good data model is difficult because all data is different, and the usage of that data varies. Generally, a smaller data model is better because it performs faster and will be simpler to use. However, defining what a smaller data model entails is equally as problematic because it's a heuristic and subjective concept.

Typically, a smaller data model is composed of fewer tables and fewer columns in each table that the user can see. If you import all necessary tables from a sales database, but the total table count is 30 tables, the user will not find that intuitive. Collapsing those tables into five tables make the data model more intuitive to the user, whereas if the user opens a table and finds 100 columns, they might find it overwhelming. Removing unneeded columns to provide a more manageable number increases the likelihood that the user reads all column names. To summarize, you should aim for simplicity when designing your data models.

The following image is an example data model. The boxes contain tables of data, where each line item within the box is a column. The lines that connect the boxes represent relationships between the tables. These relationships can be complex, even in such a simplistic model. The data model can become easily disorganized, and the total table count in the model can

gradually increase. Keeping your data model simple, comprehensive, and accurate requires constant effort.



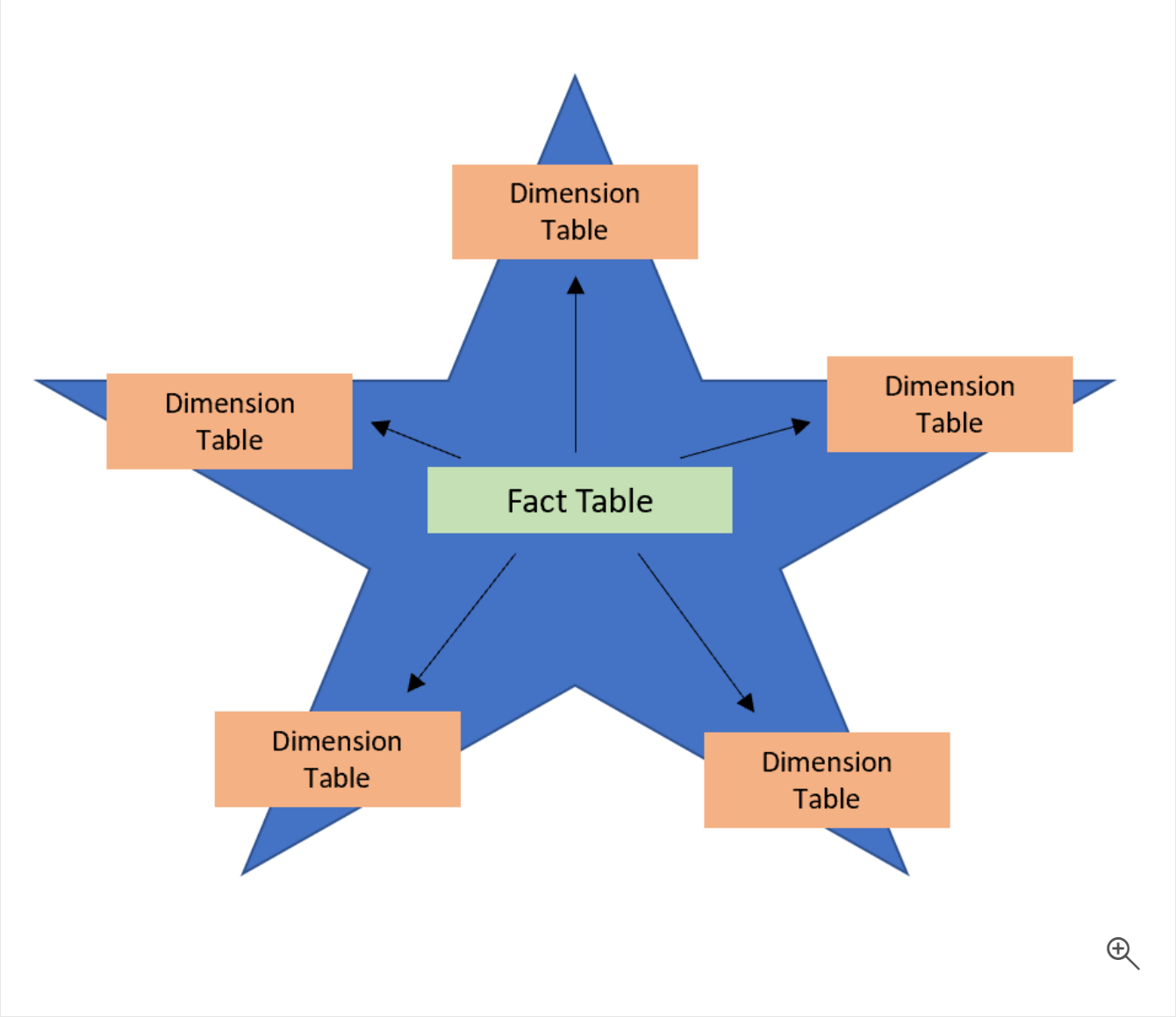
Relationships are defined between tables through primary and foreign keys. Primary keys are column(s) that identify each unique, non-null data row. For instance, if you have a Customers table, you could have an index that identifies each unique customer. The first row has an ID of 1, the second row an ID of 2, and so on. Each row is assigned a unique value, which can be referred to by this simple value: the primary key. This process becomes important when you are referencing rows in a different table, which is what foreign keys do. Relationships between tables are formed when you have primary and foreign keys in common between different tables.

Power BI allows relationships to be built from tables with different data sources, a powerful function that enables you to pull one table from Microsoft Excel and another from a relational database. You would then create the relationship between those two tables and treat them as a unified dataset.

Now that you have learned about the relationships that make up the data schema, you are able to explore a specific type of schema design, the star schema, which is optimized for high performance and usability.

Star schemas

You can design a star schema to simplify your data. It's not the only way to simplify your data, but it is a popular method; therefore, every Power BI data analyst should understand it. In a star schema, each table within your dataset is defined as a dimension or a fact table, as shown in the following visual.



Fact tables contain observational or event data values: sales orders, product counts, prices, transactional dates and times, and quantities. Fact tables can contain several repeated values. For example, one product can appear multiple times in multiple rows, for different customers on different dates. These values can be aggregated to create visuals. For instance, a visual of the total sales orders is an aggregation of all sales orders in the fact table. With fact tables, it is common to see columns that are filled with numbers and dates. The numbers can be units of measurement, such as sale amount, or they can be keys, such as a customer ID. The dates represent time that is being recorded, like order date or shipped date.

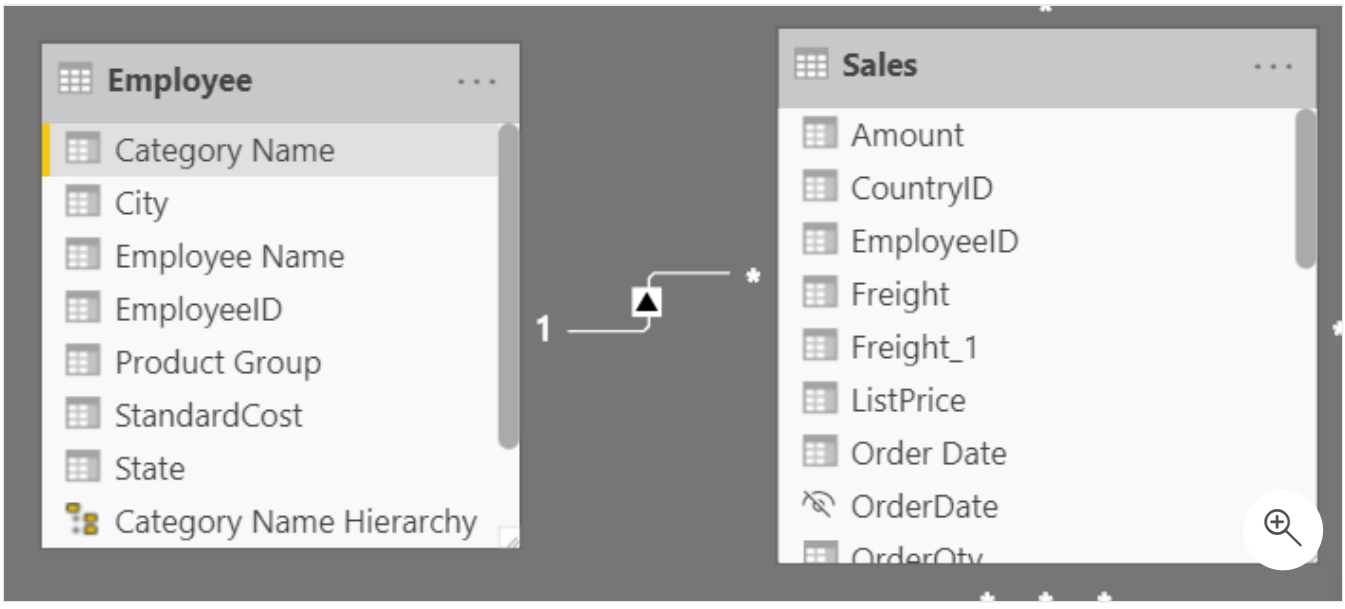
Dimension tables contain the details about the data in fact tables: products, locations, employees, and order types. These tables are connected to the fact table through key columns.

Dimension tables are used to filter and group the data in fact tables. The fact tables, on the other hand, contain the measurable data, such as sales and revenue, and each row represents a unique combination of values from the dimension tables. For the total sales orders visual, you could group the data so that you see total sales orders by product, in which product is data in the dimension table.

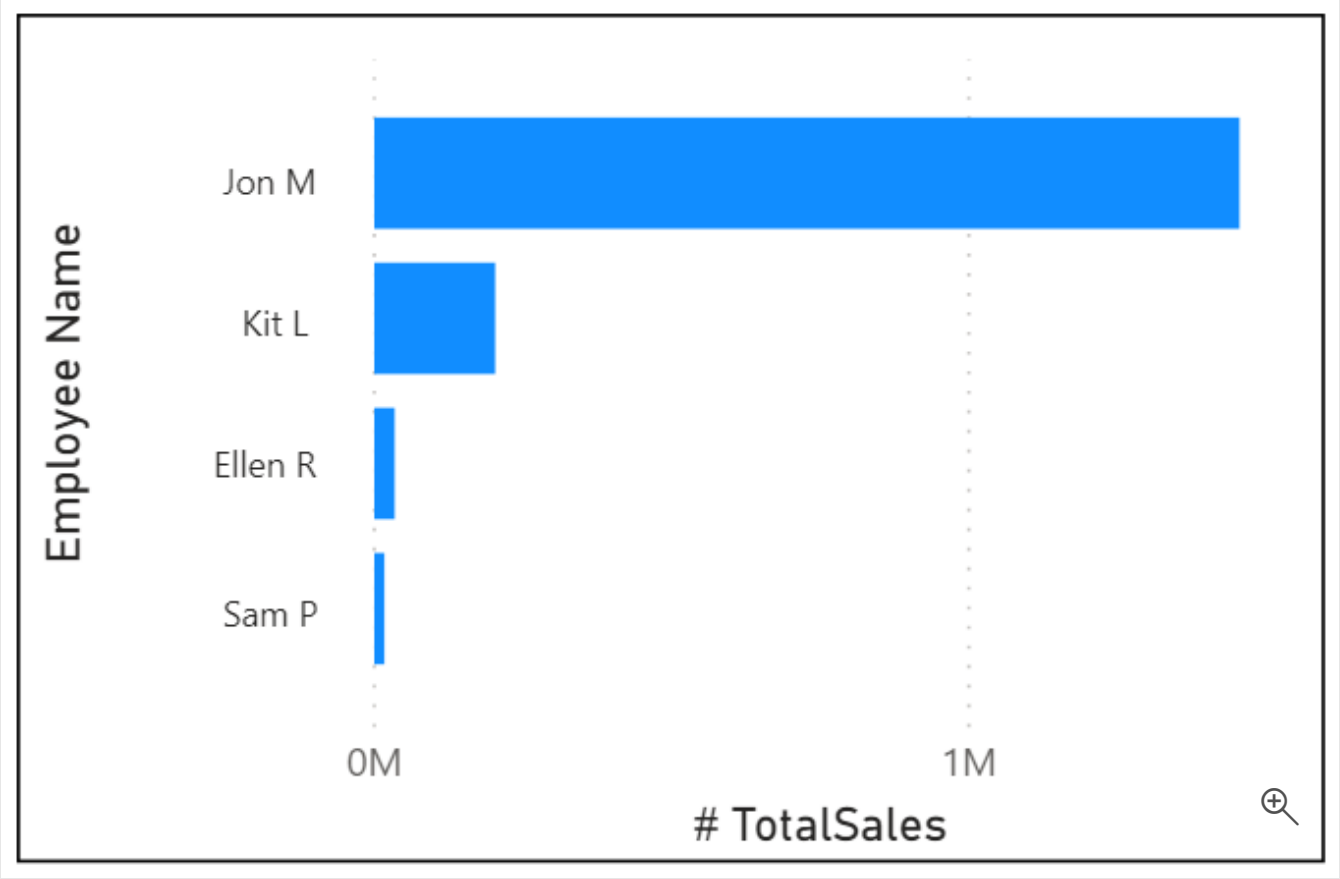
Fact tables are much larger than dimension tables because numerous events occur in fact tables, such as individual sales. Dimension tables are typically smaller because you are limited to the number of items that you can filter and group on. For instance, a year contains only so many months, and the United States are composed of only a certain number of states.

Considering this information about fact tables and dimension tables, you might wonder how you can build this visual in Power BI.

The pertinent data resides in two tables, Employee and Sales, as shown in the following data model. Because the Sales table contains the sales order values, which can be aggregated, it is considered a fact table. The Employee table contains the specific employee name, which filters the sales orders, so it would be a dimension table. The common column between the two tables, which is the primary key in the Employee table, is **EmployeeID**, so you can establish a relationship between the two tables based on this column.



When creating this relationship, you can build the visual according to the requirements, as shown in the following figure. If you did not establish this relationship, while keeping in mind the commonality between the two tables, you would have had more difficulty building your visual.



Star schemas and the underlying data model are the foundation of organized reports; the more time you spend creating these connections and design, the easier it will be to create and maintain reports.

Next unit: Work with tables

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

Work with tables

3 minutes

When users see fewer tables, they will enjoy using your data model considerably more. For example, suppose you've imported dozens of tables from many data sources and now the visual appears disorderly. In this case, you need to ensure that, before you begin working on building reports, your data model and table structure are simplified.

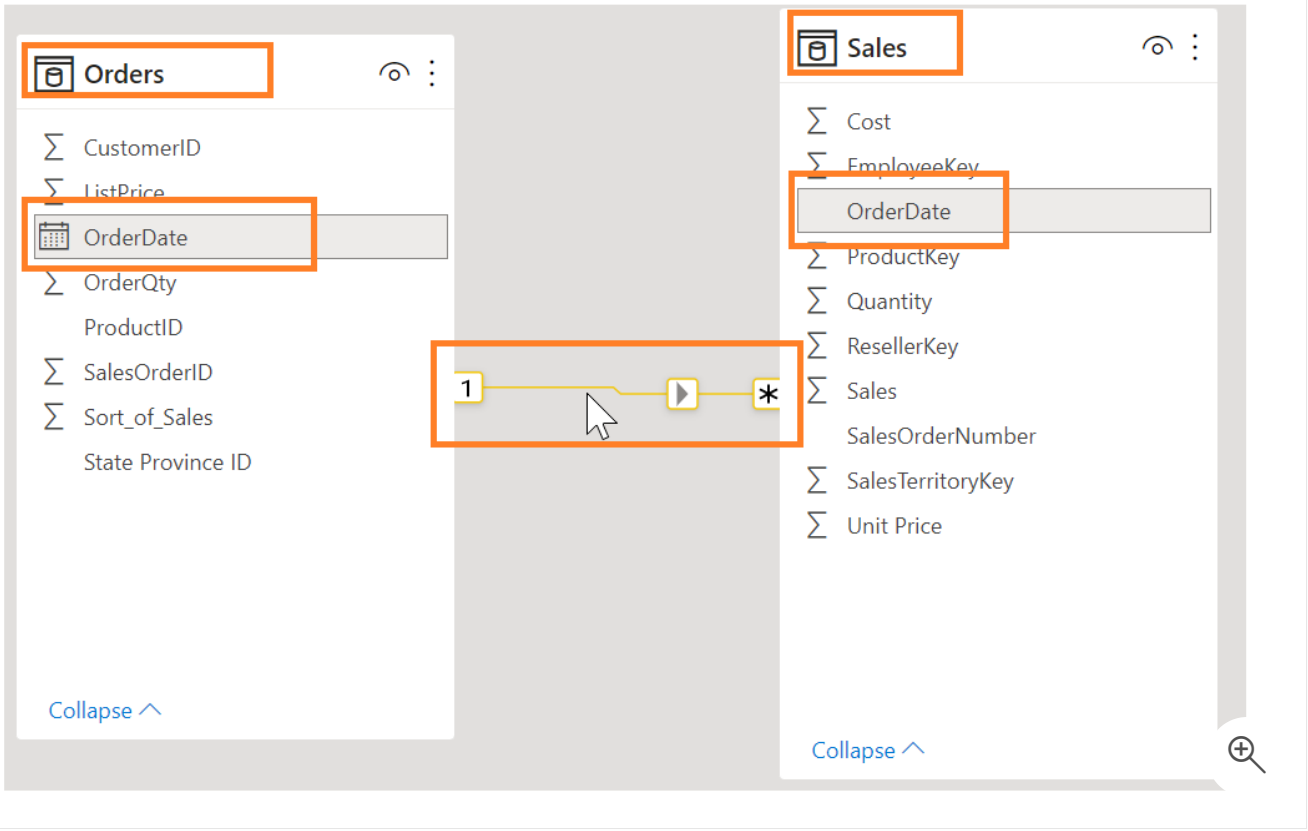
A simple table structure will:

- Be simple to navigate because of column and table properties that are specific and user-friendly.
- Have merged or appended tables to simplify the tables within your data structure.
- Have good-quality relationships between tables that make sense.

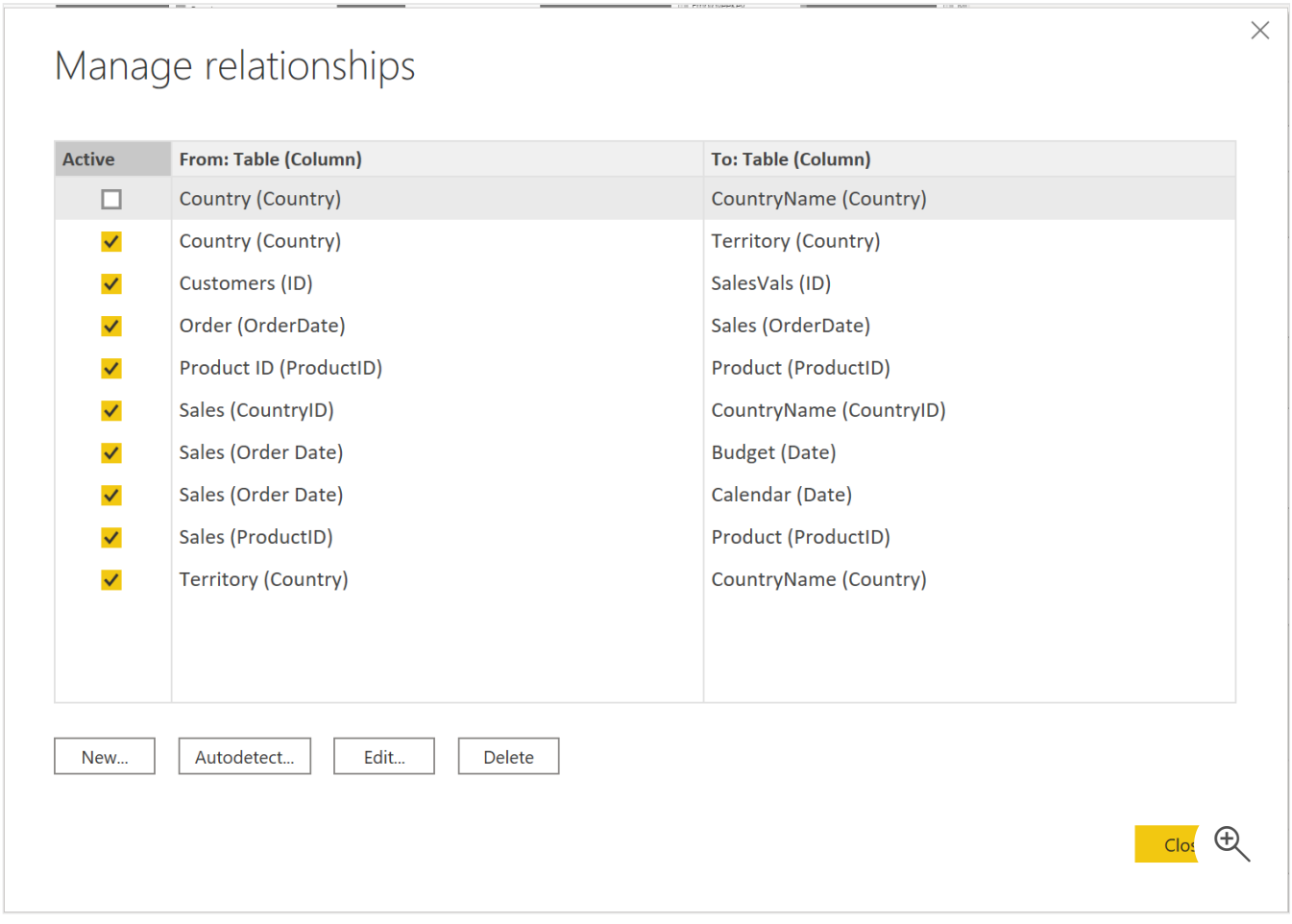
The following sections further explain how you might work with your tables to ensure a simple and readable table structure.

Configure data model and build relationships between tables

Assuming that you've already retrieved your data and cleaned it in Power Query, you can then go to the **Model** tab, where the data model is located. The following image shows how the relationship between the **Order** and **Sales** tables can be seen through the **OrderDate** column.



To manage these relationships, go to **Manage Relationships** on the ribbon, where the following window will appear.

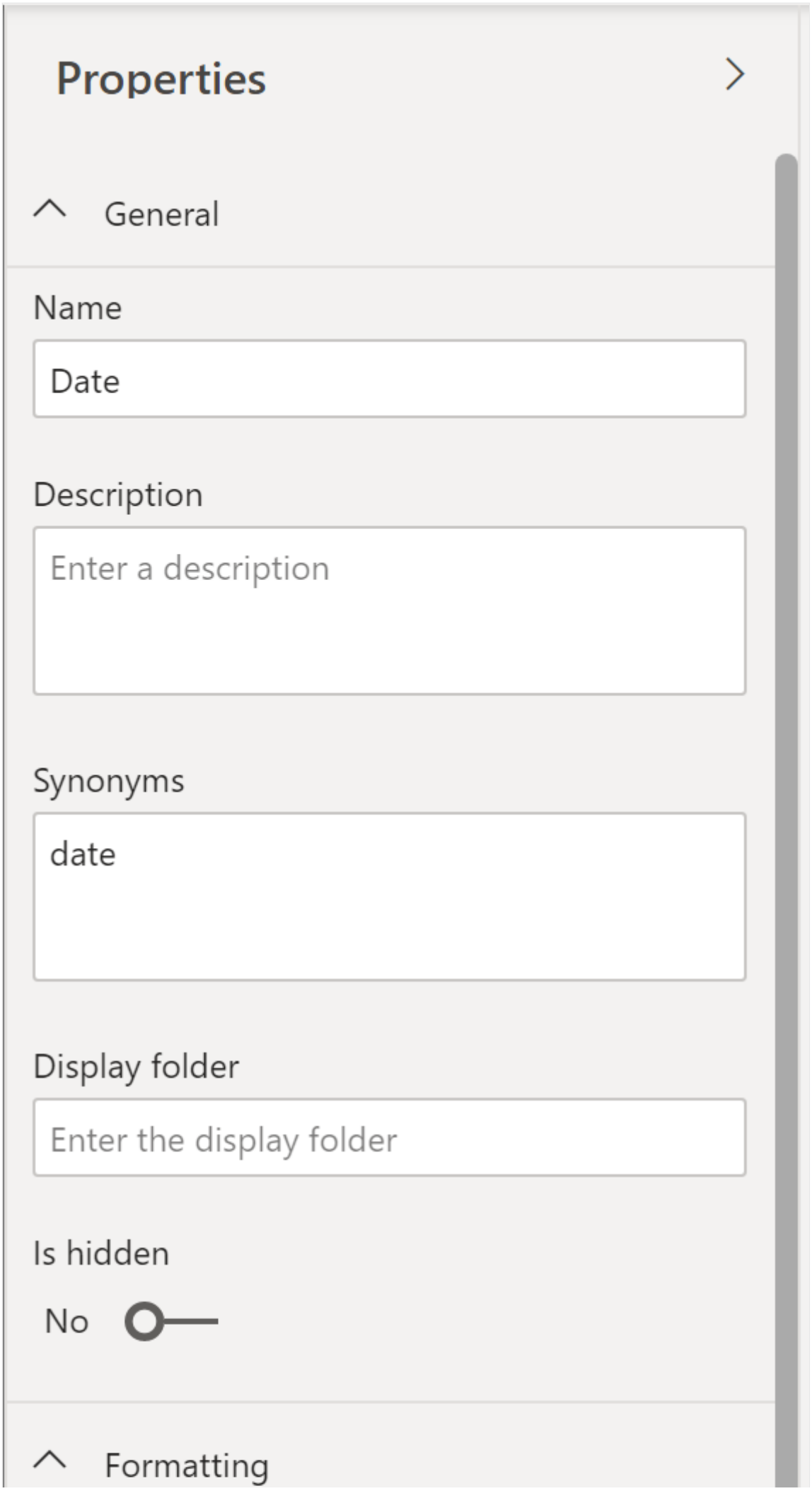


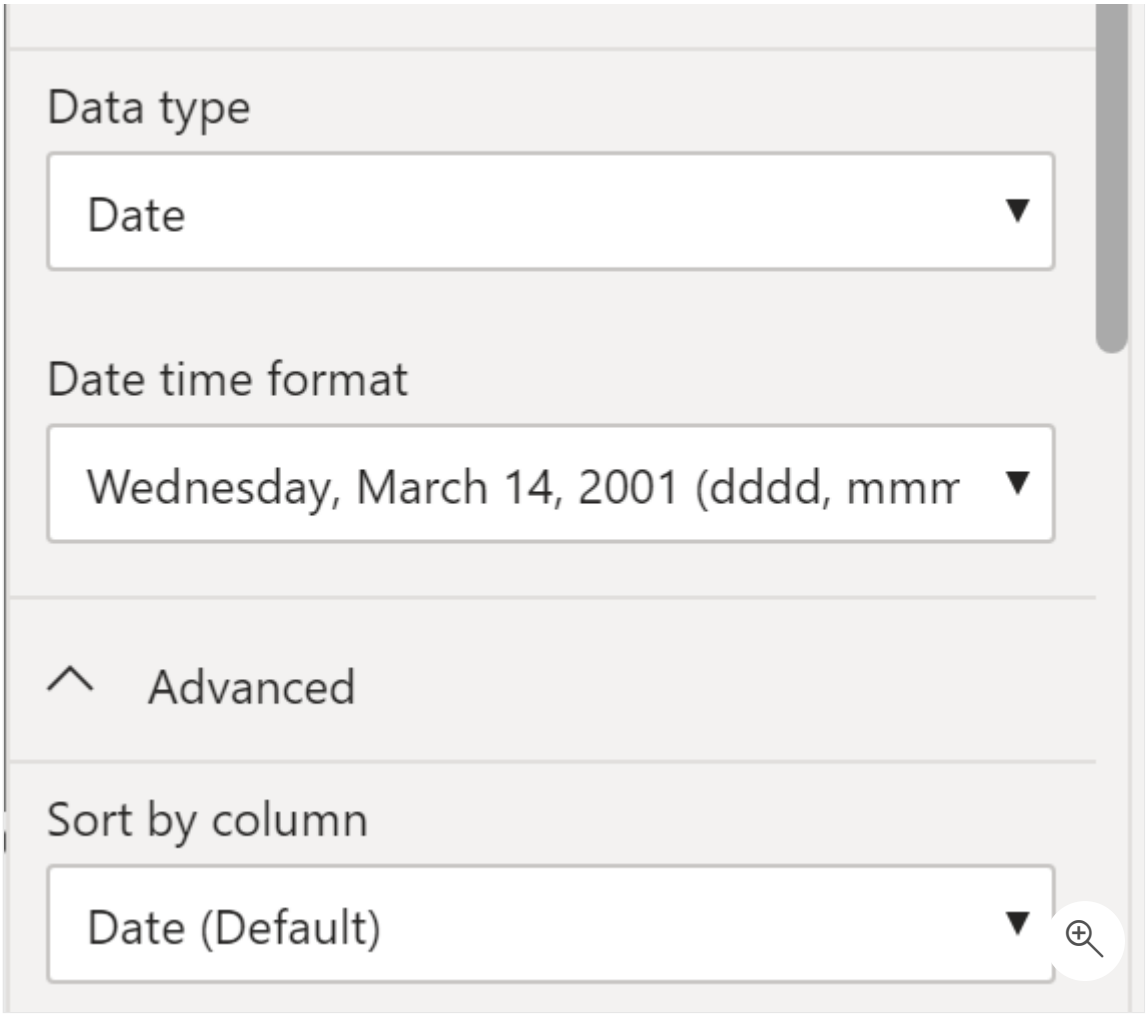
In this view, you can create, edit, and delete relationships between tables and also autodetect relationships that already exist. When you load your data into Power BI, the **Autodetect** feature will help you establish relationships between columns that are named similarly. Relationships can be inactive or active. Only one active relationship can exist between tables, which is discussed in a future module.

While the **Manage Relationships** feature allows you to configure relationships between tables, you can also configure table and column properties to ensure organization in your table structure.

Configure table and column properties

The **Model** view in Power BI desktop provides many options within the column properties that you can view or update. A simple method to get to this menu to update the tables and fields is by Ctrl+clicking or Shift+clicking items on this page.


Properties
General
Name
Date
Description
Enter a description
Synonyms
date
Display folder
Enter the display folder
Is hidden
No
Formatting



Under the **General** tab, you can:

- Edit the name and description of the column.
- Add synonyms that can be used to identify the column when you are using the Q&A feature.
- Add a column into a folder to further organize the table structure.
- Hide or show the column.

Under the **Formatting** tab, you can:

- Change the data type.
- Format the date.

For instance, suppose that the dates in your column are formatted, as seen in the previous screenshot, in the form of "Wednesday, March 14, 2001". If you want to change the format so that the date was in the "mm/dd/yyyy" format, you would select the drop-down menu under **All date time formats** and then choose the appropriate date format, as shown in the following figure.

Custom format

Custom

All date formats

*3/14/2001 (m/d/yyyy)

Wednesday, March 14, 2001 (dddd, mmmm d, yyyy)

March 14, 2001 (mmmm d, yyyy)

Wednesday, 14 March, 2001 (dddd, d mmmm, yyyy)

14 March, 2001 (d mmmm, yyyy)

3/14/2001 (m/d/yyyy)

3/14/01 (m/d/yy)

03/14/01 (mm/dd/yy)

03/14/2001 (mm/dd/yyyy)

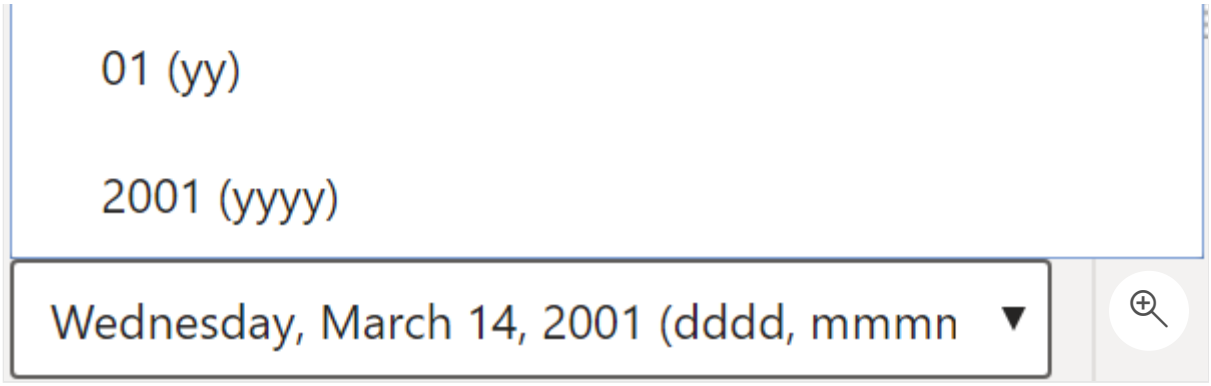
01/03/14 (yy/mm/dd)

2001-03-14 (yyyy-mm-dd)

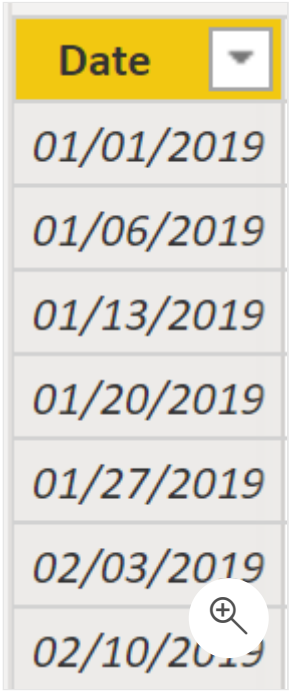
14-Mar-01 (dd-mmm-yy)

March 2001 (mmmm yyyy)

March 14 (mmmm d)



After selecting the appropriate date format, return to the **Date** column, where you should see that the format has indeed changed, as shown in the following figure.



Under the **Advanced** tab, you can:

- Sort by a specific column.
- Assign a specific category to the data.
- Summarize the data.
- Determine if the column or table contains null values.

Additionally, Power BI has a new functionality to update these properties on many tables and fields by Ctrl+clicking or Shift+clicking items.

These examples are only some of the many types of transformations that you can make to simplify the table structure. This step is important to take before you begin making your visuals so that you don't have to go back and forth when making formatting changes. This process of formatting and configuring tables can also be done in Power Query.

---

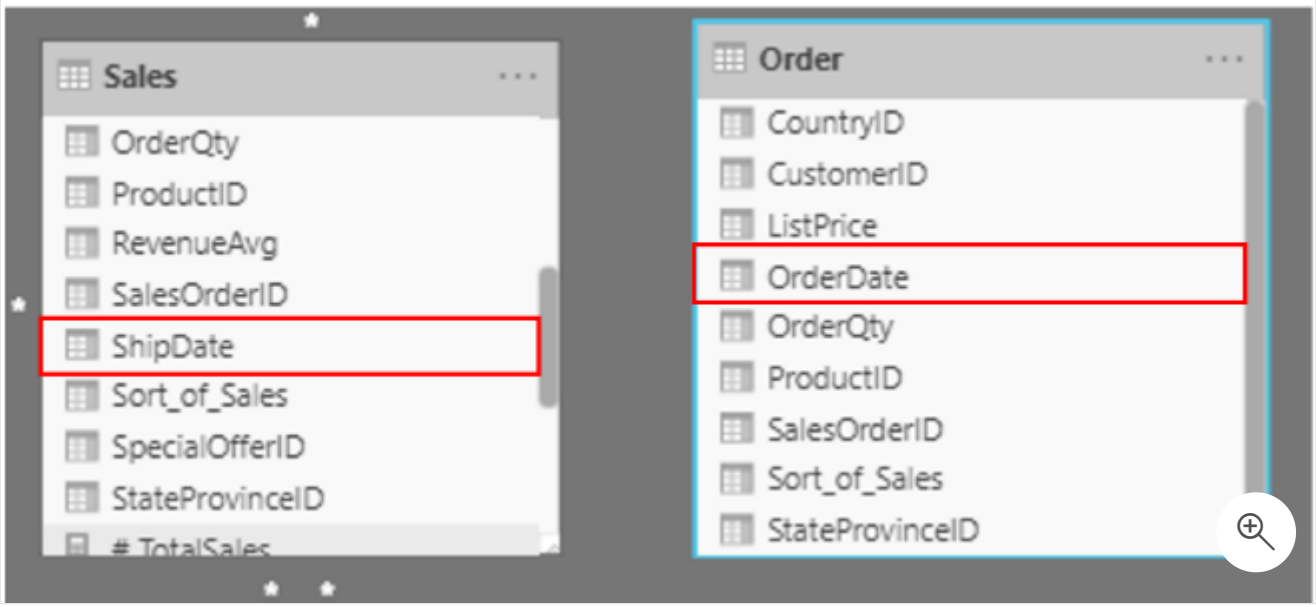
100 XP

Create a date table

9 minutes

During report creation in Power BI, a common business requirement is to make calculations based on date and time. Organizations want to know how their business is doing over months, quarters, fiscal years, and so on. For this reason, it is crucial that these time-oriented values are formatted correctly. Power BI autodetects for date columns and tables; however, situations can occur where you will need to take extra steps to get the dates in the format that your organization requires.

For example, suppose that you are developing reports for the Sales team at your organization. The database contains tables for sales, orders, products, and more. You notice that many of these tables, including Sales and Orders, contain their own date columns, as shown by the **ShipDate** and **OrderDate** columns in the Sales and Orders tables. You are tasked with developing a table of the total sales and orders by year and month. How can you build a visual with multiple tables, each referencing their own date columns?



To solve this problem, you can create a common date table that can be used by multiple tables. The following section explains how you can accomplish this task in Power BI.

Create a common date table

Ways that you can build a common date table are:

- Source data

- DAX
- Power Query

Source data

Occasionally, source databases and data warehouses already have their own date tables. If the administrator who designed the database did a thorough job, these tables can be used to perform the following tasks:

- Identify company holidays
- Separate calendar and fiscal year
- Identify weekends versus weekdays

Source data tables are mature and ready for immediate use. If you have a table as such, bring it into your data model and don't use any other methods that are outlined in this section. We recommend that you use a source date table because it is likely shared with other tools that you might be using in addition to Power BI.

If you do not have a source data table, you can use other ways to build a common date table.

DAX

You can use the Data Analysis Expression (DAX) functions CALENDARAUTO() or CALENDAR() to build your common date table. The CALENDAR() function returns a contiguous range of dates based on a start and end date that are entered as arguments in the function. Alternatively, the CALENDARAUTO() function returns a contiguous, complete range of dates that are automatically determined from your dataset. The starting date is chosen as the earliest date that exists in your dataset, and the ending date is the latest date that exists in your dataset plus data that has been populated to the fiscal month that you can choose to include as an argument in the CALENDARAUTO() function. For the purposes of this example, the CALENDAR() function is used because you only want to see the data from May 31, 2011 (the first day that Sales began its tracking of this data) and forward for the next 10 years.

In Power BI Desktop, go to the **Table** tab on the ribbon. Select **New Table**, and then enter in the following DAX formula:

DAX

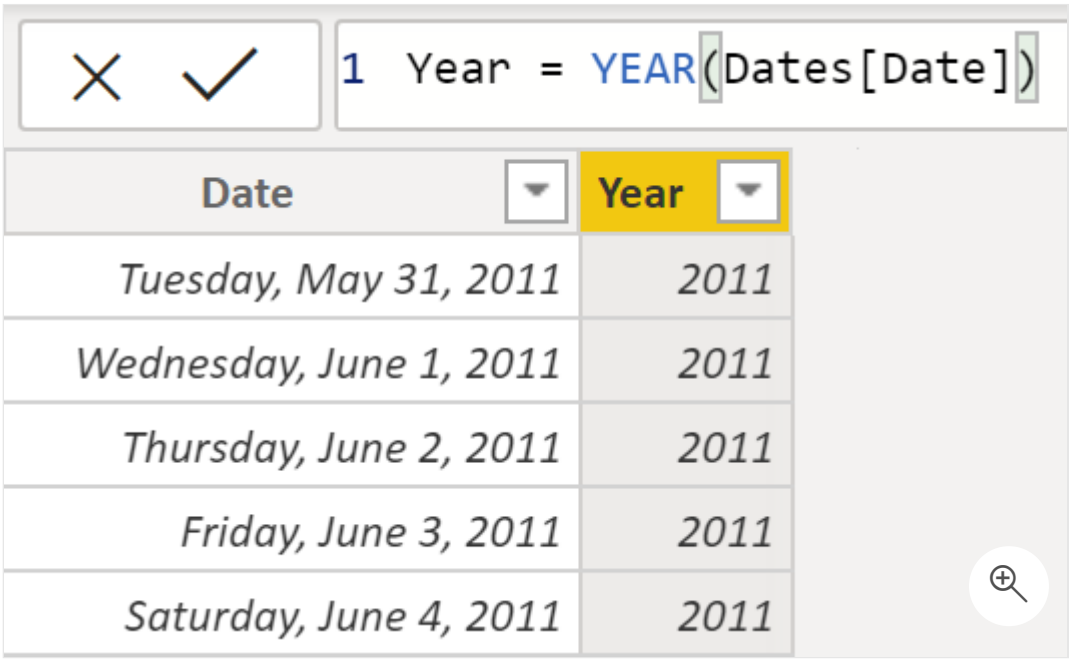
```
Dates  = CALENDAR(DATE(2011, 5, 31), DATE(2022, 12, 31))
```

```
1 Dates = CALENDAR(DATE(2011, 5, 31), DATE(2022, 12, 31
```

Now, you have a column of dates that you can use. However, this column is slightly sparse. You also want to see columns for just the year, the month number, the week of the year, and the day of the week. You can accomplish this task by selecting **New Column** on the ribbon and entering the following DAX equation, which will retrieve the year from your Date table.

DAX

```
Year = YEAR(Dates[Date])
```



You can perform the same process to retrieve the month number, week number, and day of the week:

DAX

```
MonthNum = MONTH(Dates[Date])
```

DAX

```
WeekNum = WEEKNUM(Dates[Date])
```

DAX

```
DayoftheWeek = FORMAT(Dates[Date], "DDDD")
```

When you have finished, your table will contain the columns that are shown in the following figure.

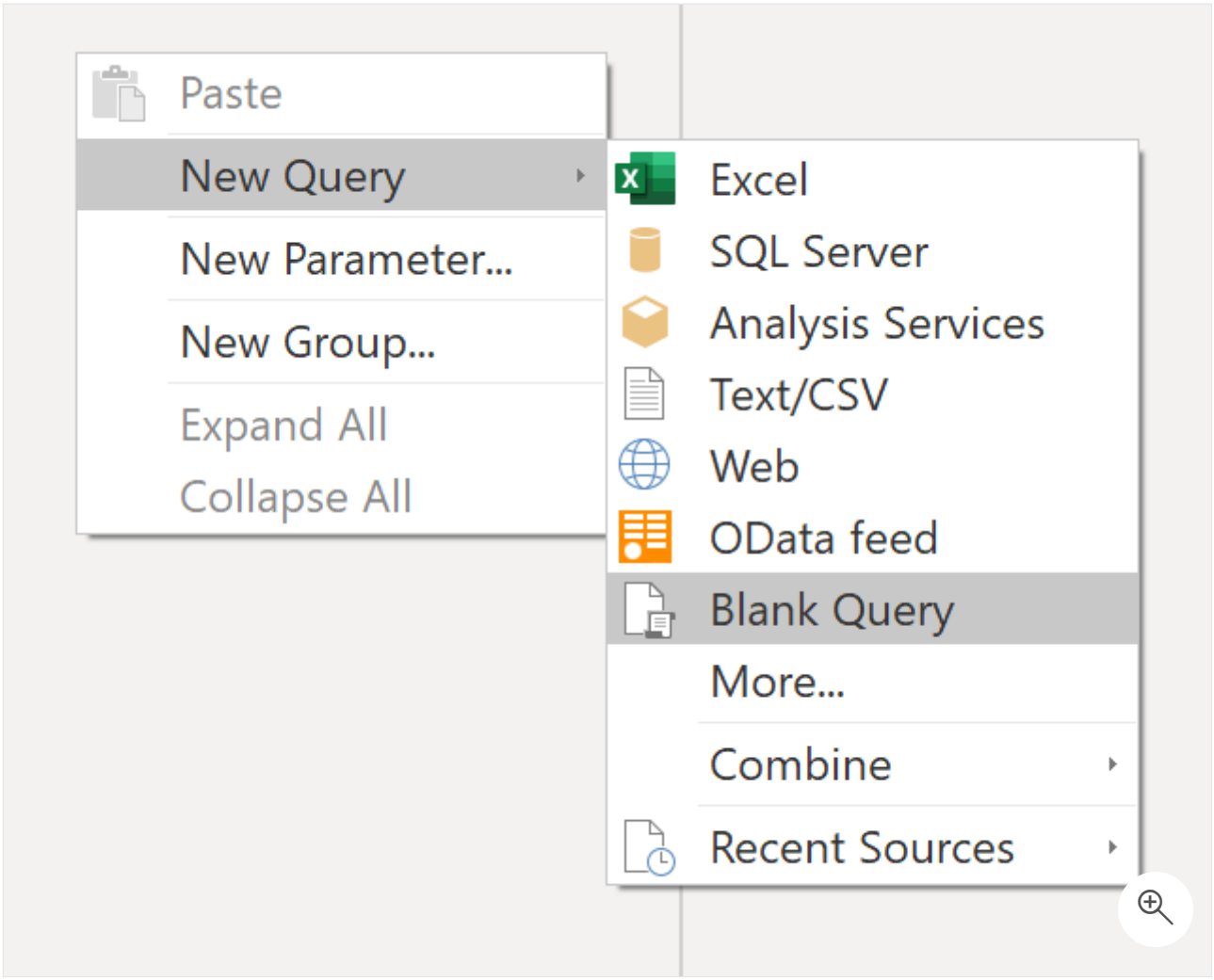
Date	Year	MonthNum	WeekNum	DayoftheWeek
Tuesday, May 31, 2011	*2011*	*5*	*23*	Tuesday
Wednesday, June 1, 2011	*2011*	*6*	*23*	Sunday
Thursday, June 2, 2011	*2011*	*6*	*23*	Monday
Friday, June 3, 2011	*2011*	*6*	*23*	Tuesday

You have now created a common date table by using DAX. This process only adds your new table to the data model; you will still need to establish relationships between your date table and the Sales and Order tables, and then mark your table as the official date table of your data model. However, before you complete those tasks, make sure that you consider another way of building a common date table: by using Power Query.

Power Query

You can use M-language, the development language that is used to build queries in Power Query, to define a common date table.

Select **Transform Data** in Power BI Desktop, which will direct you to Power Query. In the blank space of the left **Queries** pane, right-click to open the following drop-down menu, where you will select **New Query** > **Blank Query**.



In the resulting **New Query** view, enter the following M-formula to build a calendar table:

M

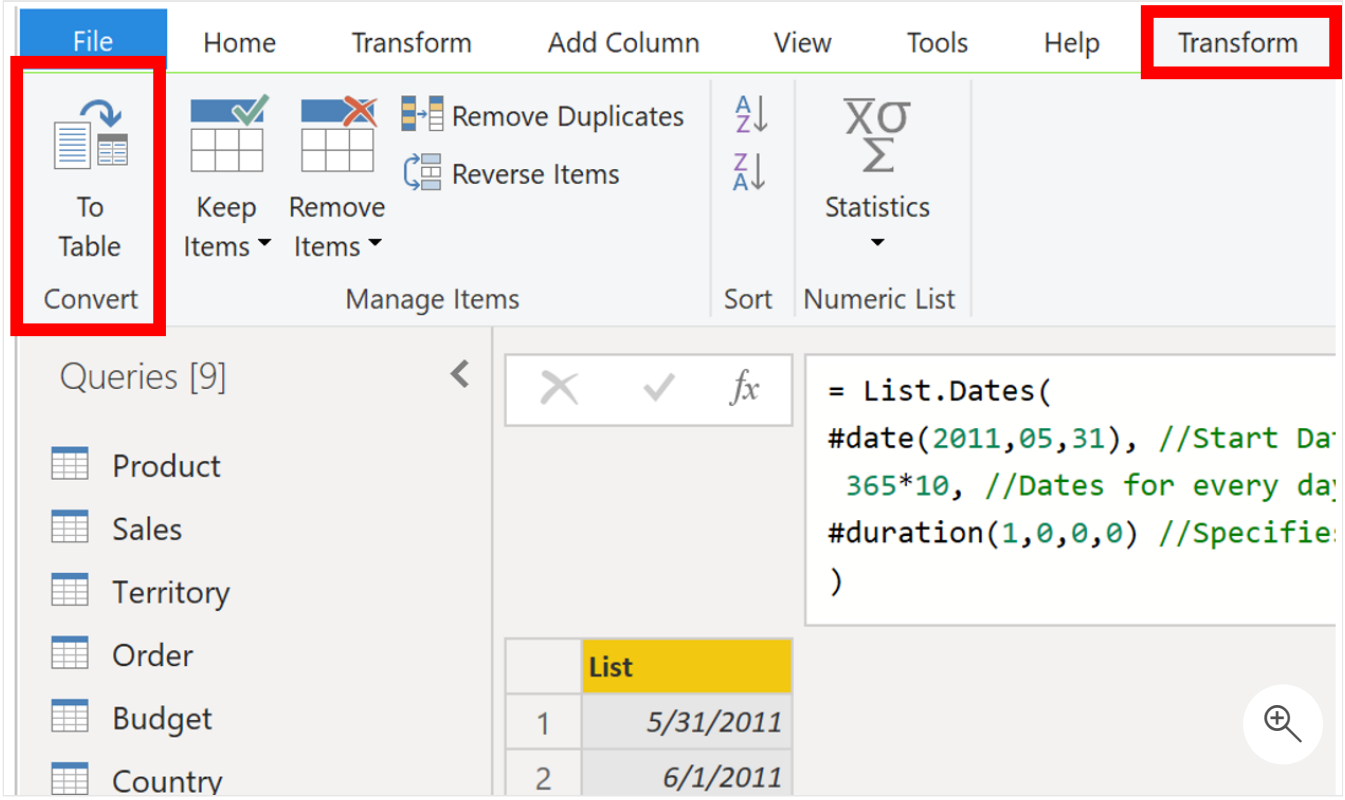
```
= List.Dates(#date(2011,05,31), 365*10, #duration(1,0,0,0))
```

```
= List.Dates(
#date(2011,05,31), //Start Date
 365*10, //Dates for every day for the next 10 years
#duration(1,0,0,0) //Specifies duration of the period 1 = days, 0 = hours, 0 = minutes, 0 = s
)
```

For your sales data, you want the start date to reflect the earliest date that you have in your data: May 31, 2011. Additionally, you want to see dates for the next 10 years, including dates in the future. This approach ensures that, as new sales data flows in, you won't have to re-create this table. You can also change duration. In this case, you want a data point for every day, but you can also increment by hours, minutes, and seconds. The following figure shows the result.

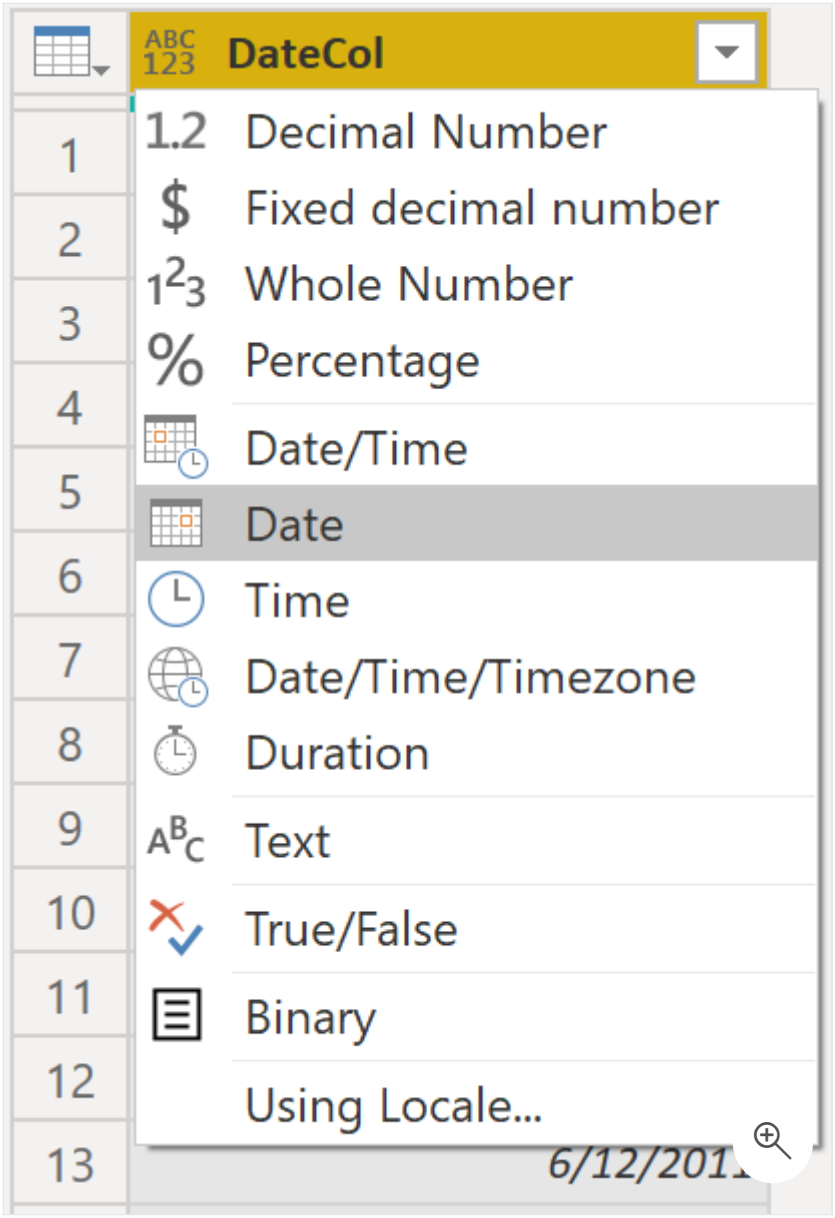
	List
1	5/31/2011
2	6/1/2011
3	6/2/2011
4	6/3/2011
5	6/4/2011
6	6/5/2011
7	6/6/2011
8	6/7/2011
9	6/8/2(

After you have realized success in the process, you notice that you have a list of dates instead of a table of dates. To correct this error, go to the **Transform** tab on the ribbon and select **Convert** > **To Table**. As the name suggests, this feature will convert your list into a table. You can also rename the column to **DateCol**.

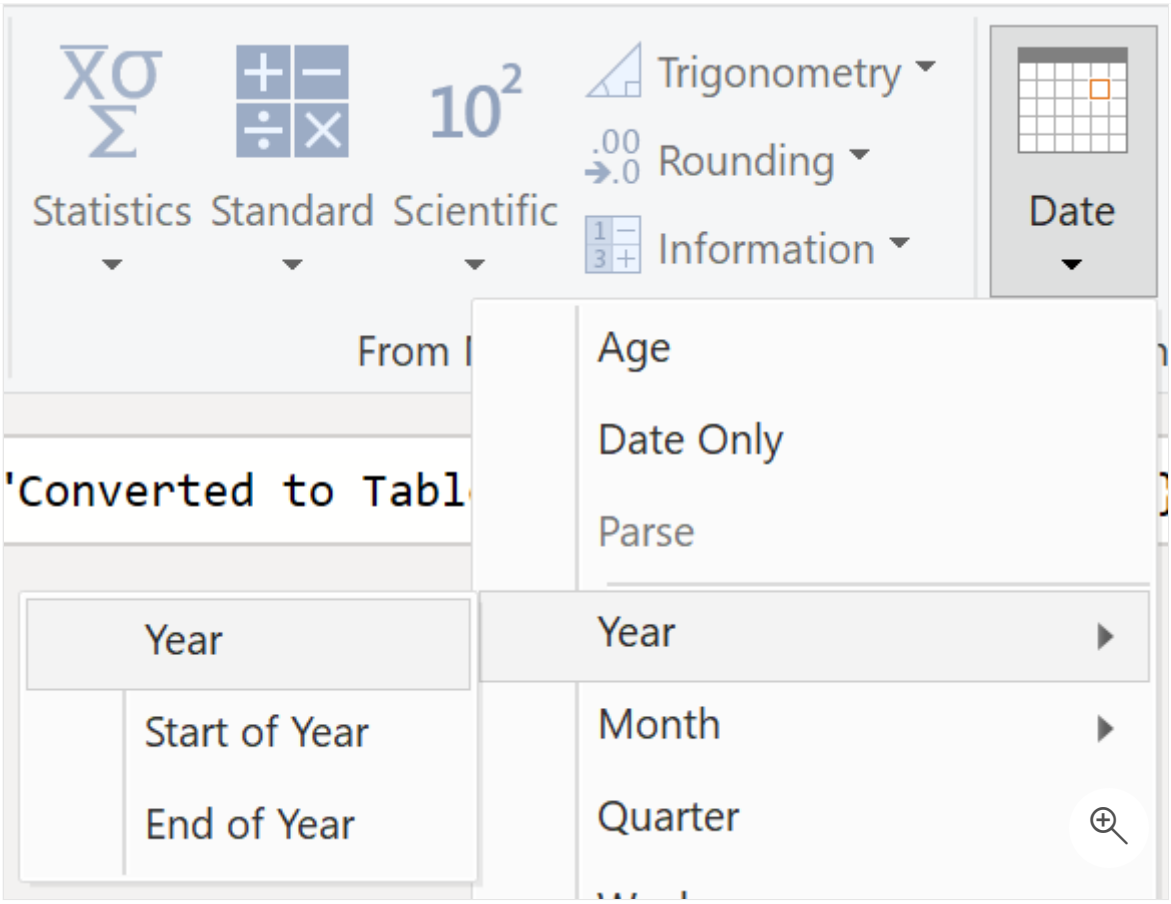


Next, you want to add columns to your new table to see dates in terms of year, month, week, and day so that you can build a hierarchy in your visual. Your first task is to change the column

type by selecting the icon next to the name of the column and, in the resulting drop-down menu, selecting the **Date** type.



After you have finished selecting the **Date** type, you can add columns for year, months, weeks, and days. Go to **Add Column**, select the drop-down menu under **Date**, and then select **Year**, as shown in the following figure.



Notice that Power BI has added a column of all years that are pulled from **DateCol**.

DateCol	Year
5/31/2011	2011
6/1/2011	2011
6/2/2011	2011

Complete the same process for months, weeks, and days. After you have finished this process, the table will contain the columns that are shown in the following figure.

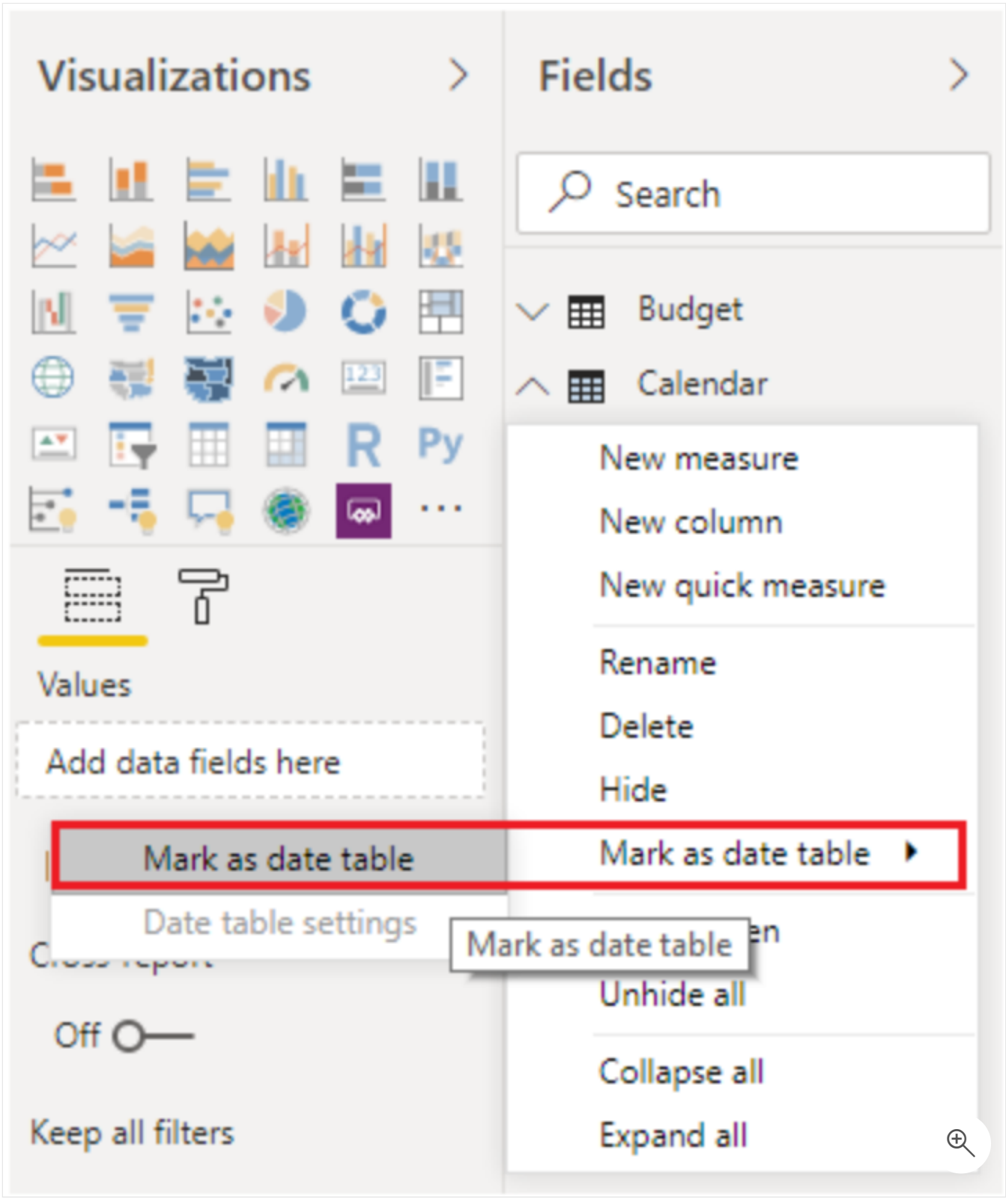
	DateCol	Year	Month	Week of Year	Day Name
1	5/31/2011	2011	5	23	Tuesday
2	6/1/2011	2011	6	23	Wednesday
3	6/2/2011	2011	6	23	Thursday

You have now successfully used Power Query to build a common date table.

The previous steps show how to get the table into the data model. Now, you need to mark your table as the official date table so that Power BI can recognize it for all future values and ensure that formatting is correct.

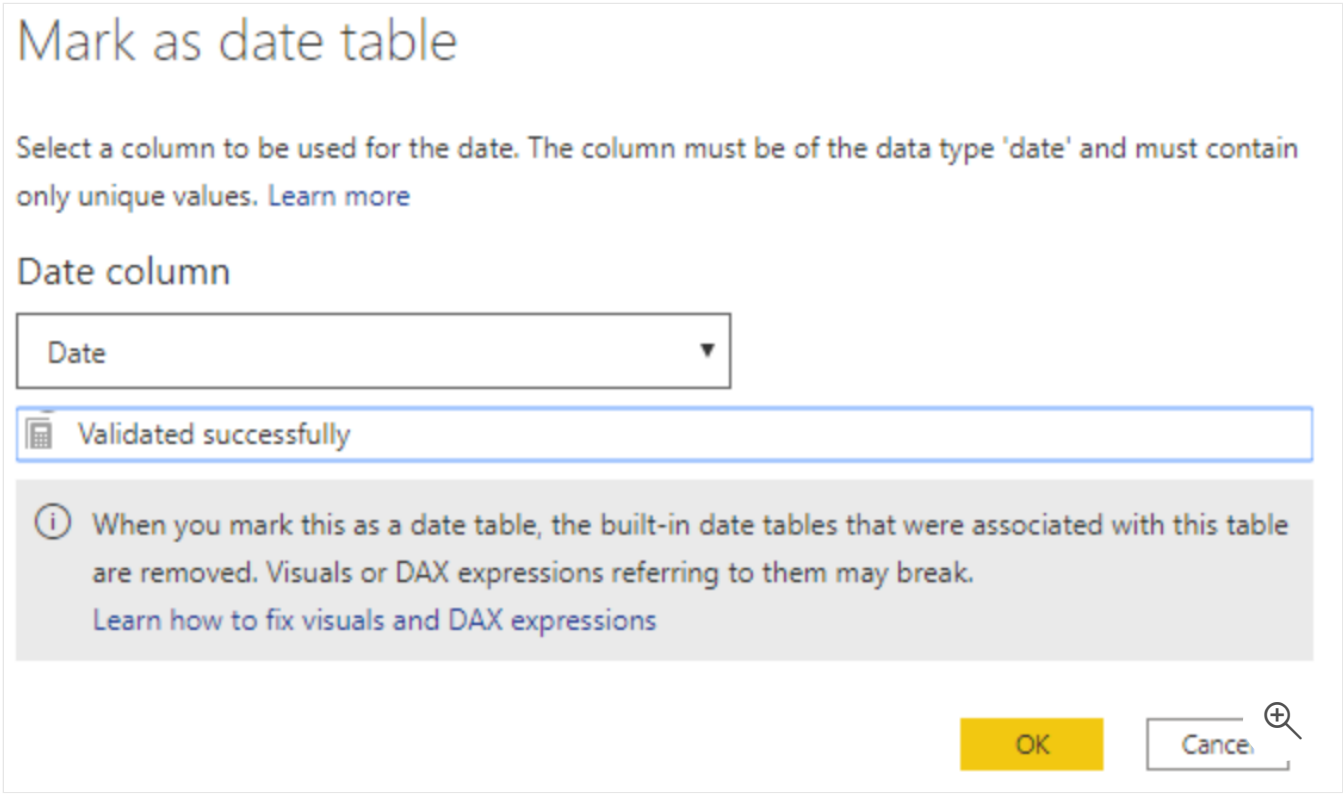
Mark as the official date table

Your first task in marking your table as the official date table is to find the new table on the **Fields** pane. Right-click the name of the table and then select **Mark as date table**, as shown in the following figure.



By marking your table as a date table, Power BI performs validations to ensure that the data contains zero null values, is unique, and contains continuous date values over a period. You can also choose specific columns in your table to mark as the date, which can be useful when you

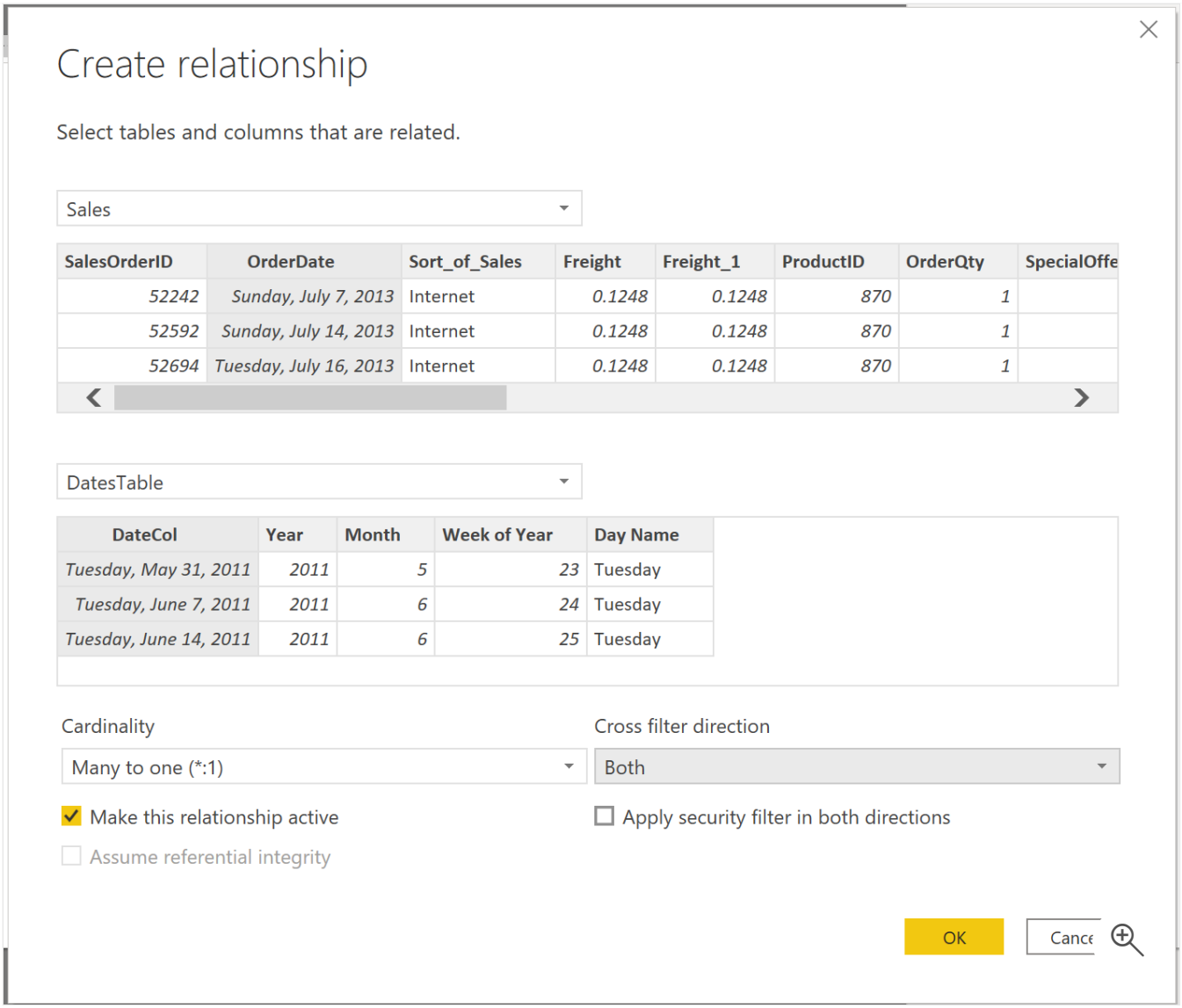
have many columns within your table. Right-click the table, select **Mark as date table**, and then select **Date table settings.** The following window will appear, where you can choose which column should be marked as **Date**.



Selecting **Mark as date table** will remove autogenerated hierarchies from the **Date** field in the table that you marked as a date table. For other date fields, the auto hierarchy will still be present until you establish a relationship between that field and the date table or until you turn off the **Auto Date/Time** feature. You can manually add a hierarchy to your common date table by right-clicking the year, month, week, or day columns in the **Fields** pane and then selecting **New hierarchy.** This process is further discussed later in this module.

Build your visual

To build your visual between the Sales and Orders tables, you will need to establish a relationship between this new common date table and the Sales and Orders tables. As a result, you will be able to build visuals by using the new date table. To complete this task, go to **Model** tab > **Manage Relationships**, where you can create relationships between the common date table and the Orders and Sales tables by using the **OrderDate** column. The following screenshot shows an example of one such relationship.



After you have built the relationships, you can build your **Total Sales and Order Quantity by Time** visual with your common date table that you developed by using the DAX or Power Query method.

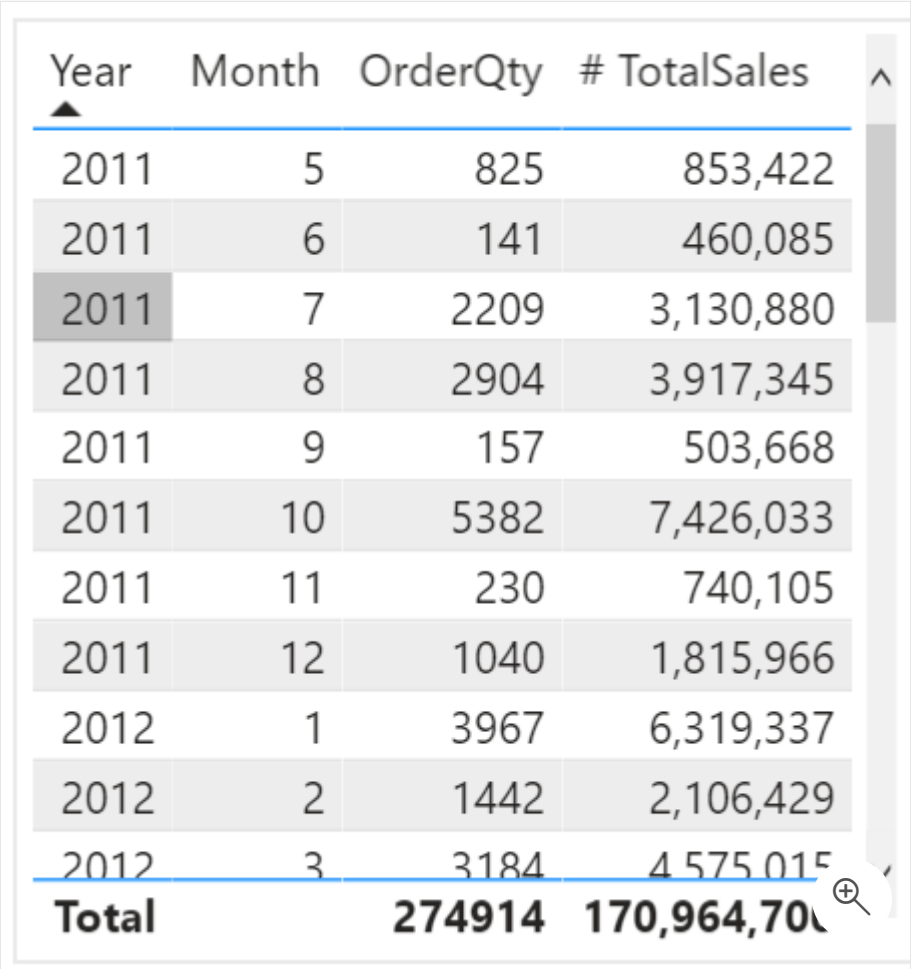
To determine the total sales, you need to add all sales because the **Amount** column in the Sales table only looks at the revenue for each sale, not the total sales revenue. You can complete this task by using the following measure calculation, which will be explained in later discussions. The calculation that you will use when building this measure is as follows:

DAX

```
#Total Sales = SUM(Sales['Amount'])
```

After you have finished, you can create a table by returning to the **Visualizations** tab and selecting the **Table** visual. You want to see the total orders and sales by year and month, so you only want to include the Year and Month columns from your date table, the **OrderQty** column, and the **#TotalSales** measure. When you learn about hierarchies, you can also build a

hierarchy that will allow you drill down from years to months. For this example, you can view them side-by-side. You have now successfully created a visual with a common date table.



Year	Month	OrderQty	# TotalSales
2011	5	825	853,422
2011	6	141	460,085
2011	7	2209	3,130,880
2011	8	2904	3,917,345
2011	9	157	503,668
2011	10	5382	7,426,033
2011	11	230	740,105
2011	12	1040	1,815,966
2012	1	3967	6,319,337
2012	2	1442	2,106,429
2012	3	3184	4,575,015
Total		**274914**	**170,964,70**

Next unit: Work with dimensions

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Work with dimensions

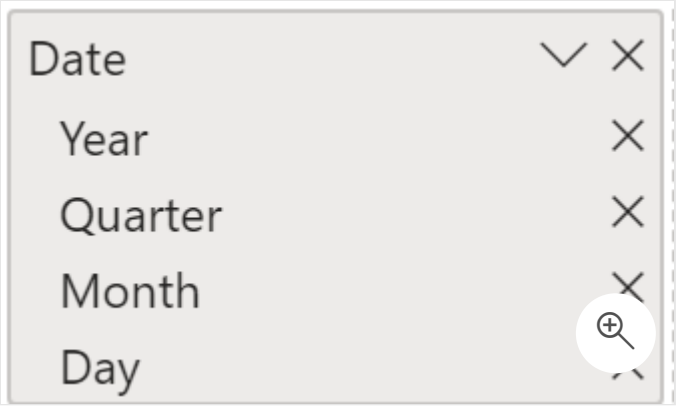
6 minutes

When building a star schema, you will have dimension and fact tables. Fact tables contain information about events such as sales orders, shipping dates, resellers, and suppliers. Dimension tables store details about business entities, such as products or time, and are connected back to fact tables through a relationship.

You can use hierarchies as one source to help you find detail in dimension tables. These hierarchies form through natural segments in your data. For instance, you can have a hierarchy of dates in which your dates can be segmented into years, months, weeks, and days. Hierarchies are useful because they allow you to drill down into the specifics of your data instead of only seeing the data at a high level.

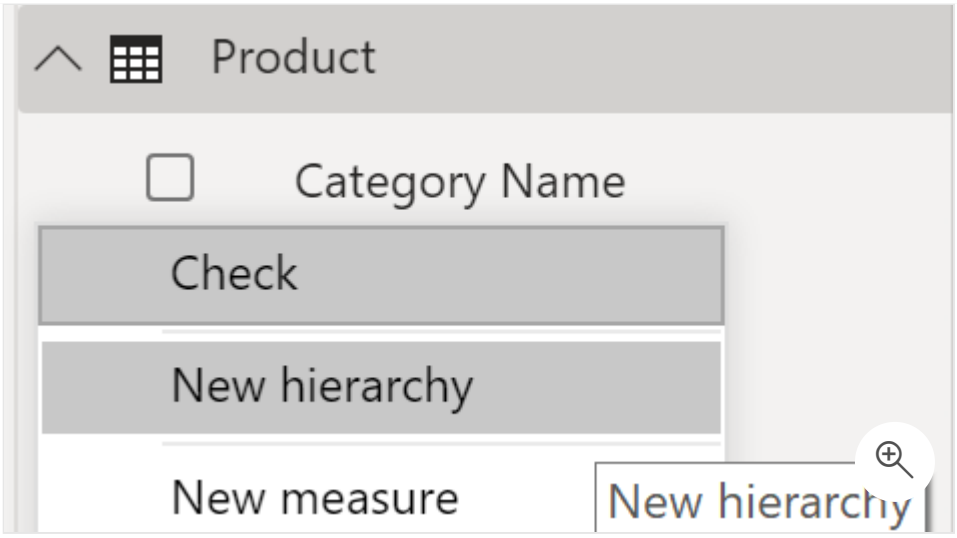
Hierarchies

When you are building visuals, Power BI automatically enters values of the date type as a hierarchy (if the table has not been marked as a date table).

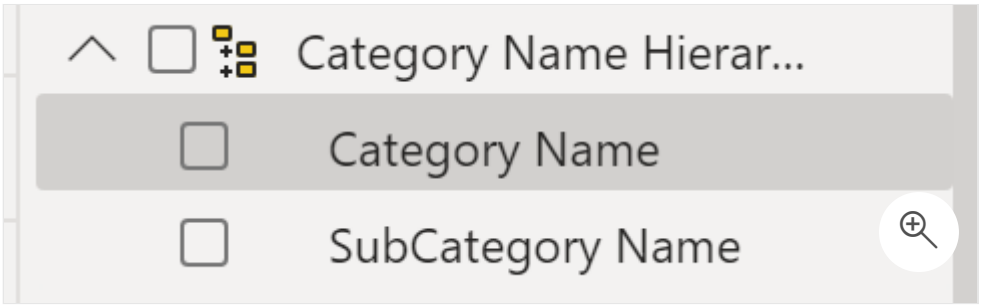


In the preceding **Date** column, the date is shown in increasingly finer detail through year, quarters, months, and days. You can also manually create hierarchies.

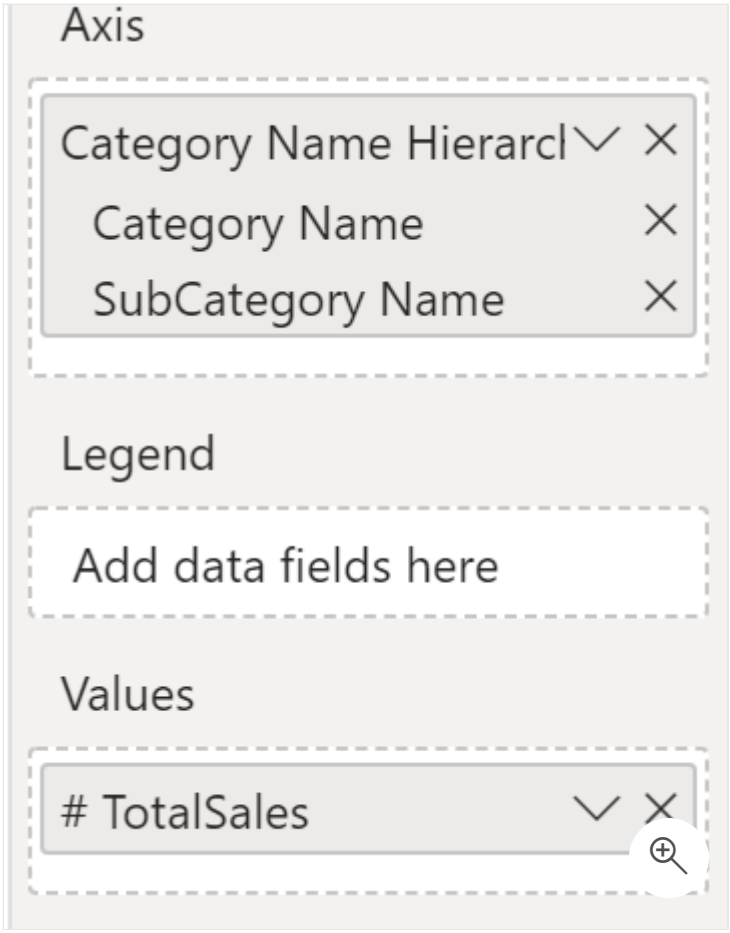
For example, consider a situation where you want to create a stacked bar chart of **Total Sales by Category and Subcategory**. You can accomplish this task by creating a hierarchy in the **Product** table for categories and subcategories. To create a hierarchy, go to the **Fields** pane on Power BI and then right-click the column that you want the hierarchy for. Select **New hierarchy**, as shown in the following figure.



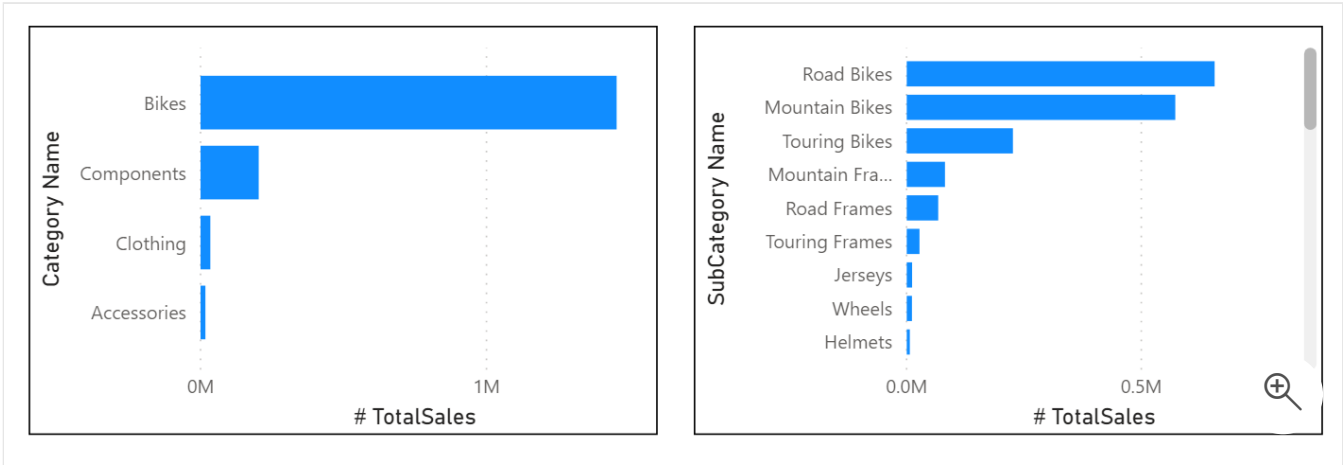
Next, drag and drop the subcategory column into this new hierarchy that you've created. This column will be added as a sublevel on the hierarchy.



Now, you can build the visual by selecting a stacked bar chart in the **Visualizations** pane. Add your **Category Name Hierarchy** in the **Axis** field and **Total Sales** in the **Values** field.



You can drill down on the visual to view both **Category** and **Subcategory**, depending on what you want to see. Hierarchies allow you to view increasing levels of data on a single view.



Now that you have learned about hierarchies, you can take a step further and examine parent-child hierarchies and their role in multiple relationships between fact tables and dimension tables.

Parent-child hierarchy

In the following example, you have an Employee table within the database that tells you important information about the employees, their managers, and their IDs. When looking at this table, you notice that **Roy F** has been repeated multiple times in the **Manager** column. As

the image shows, multiple employees can have the same manager, which indicates a hierarchy between managers and employees.

	Employee ID	Employee	Manager ID	Manager
1	1010	Roy F	null	
2	1011	Pam H	1010	Roy F
3	1012	Guy L	1010	Roy F
4	1013	Roger M	1011	Pam H
5	1014	Kaylie S	1011	Pam H
6	1015	Mike O	1012	Guy L
7	1016	Rudy Q	1012	Guy L

The **Manager** column determines the hierarchy and is therefore the parent, while the "children" are the employees. For this example, you want to be able to see all levels of this hierarchy. Power BI does not default to showing you all levels of the hierarchy, so it is your responsibility to ensure that you see all levels of this hierarchy or "flatten" it so that you can see more data granularity.

Flatten parent-child hierarchy

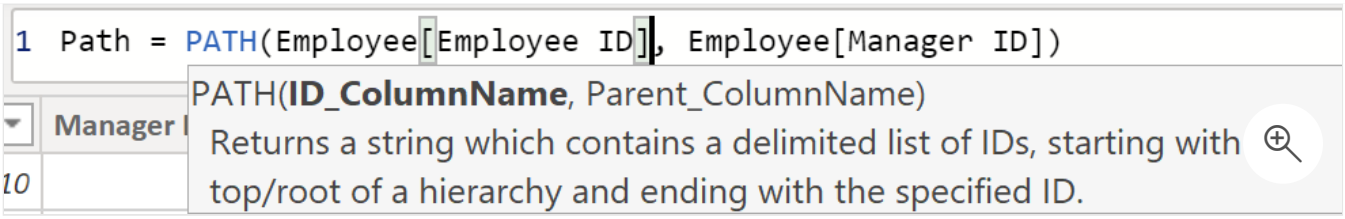
The process of viewing multiple child levels based on a top-level parent is known as *flattening the hierarchy*. In this process, you are creating multiple columns in a table to show the hierarchical path of the parent to the child in the same record. You will use PATH(), a simple DAX function that returns a text version of the managerial path for each employee, and PATHITEM() to separate this path into each level of managerial hierarchy.

> **Important**
>
> DAX has not been covered yet; however, it will be in another module. This function is included in this section because it's explaining hierarchies. If use of DAX in this capacity is confusing, refer to the DAX module and then return to this section afterward.

While on the table, go to the **Modeling** tab and select **New Column**. In the resulting formula bar, enter the following function, which creates the text path between the employee and manager. This action creates a calculated column in DAX.

DAX

```
Path = PATH(Employee[Employee ID], Employee[Manager ID])
```



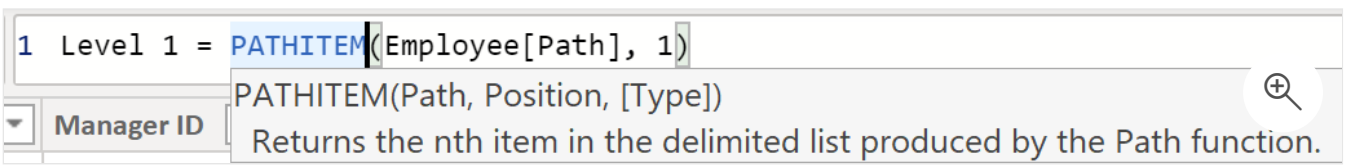
The completed path between the employee and the manager appears in the new column, as shown in the following screenshot.

Employee ID	Manager ID	Employee	Manager	Path
1010		Roy F		1010
1011	1010	Pam H	Roy F	1010|1011
1012	1010	Guy L	Roy F	1010|1012
1013	1011	Roger M	Pam H	1010|1011|1013
1014	1011	Kaylie S	Pam H	1010|1011|1014
1015	1012	Mike O	Guy L	1010|1012|...
1016	1012	Rudy Q	Guy L	1010|1012|1016

If you look at Roger M, the path of IDs is **1010 | 1011 | 1013**, which means that one level above Roger M (ID 1013) is his manager, Pam H (ID 1011), and one level above Pam H is her manager Roy F (ID 1010). In this row, Roger M is on the bottom of the hierarchy, at the child level, and Roy F is at the top of the hierarchy and is at the parent level. This path is created for every employee. To flatten the hierarchy, you can separate each level by using the PATHITEM function.

To view all three levels of the hierarchy separately, you can create four columns in the same way that you did previously, by entering the following equations. You will use the PATHITEM function to retrieve the value that resides in the corresponding level of your hierarchy.

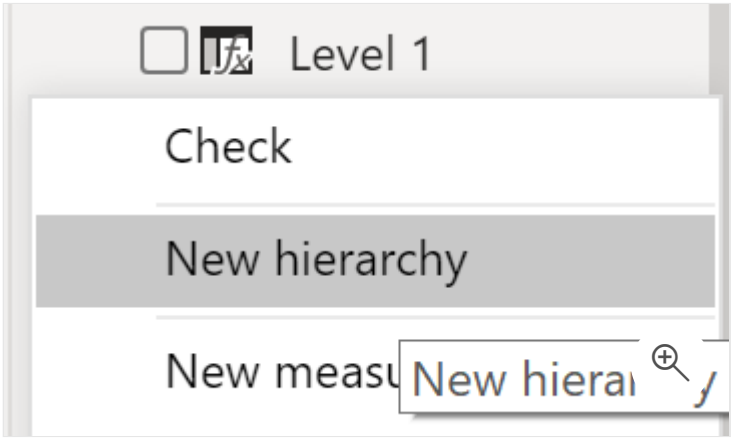
- *Level 1 = PATHITEM(Employee[Path],1)*
- *Level 2 = PATHITEM(Employee[Path],2)*
- *Level 3 = PATHITEM(Employee[Path],3)*



After you have finished, notice that you now have each level of hierarchy within your table. Roy F is at the top of the hierarchy and, as you go through **Levels 2-3**, notice that the managers and employees map with each other.

Employee ID	Name	Manager	Manager ID	Path	Level 1	Level 2	Level 3	Level 4
1000	Quincy Howard			1000	1000			
1001	Mallory Yang	Quincy Howard	1000	1000|1001	1000	1001		
1002	Donovan Maynard	Quincy Howard	1000	1000|1002	1000	1002		
1003	Giselle Mcclain	Mallory Yang	1001	1000|1001|1003	1000	1001	1003	
1004	Melvin Marsh	Mallory Yang	1001	1000|1001|1004	1000	1001	1004	
1005	Ria Snow	Giselle Mcclain	1003	1000|1001|1003|1005	1000	1001	1003	10
1006	Callie Savage	Giselle Mcclain	1003	1000|1001|1003|1006	1000	1001	1003	1006

Now, you can create a hierarchy on the **Fields** pane, as you did previously. Right-click **Level 1**, because this is the first hierarchy level, and then select **New Hierarchy.** Then, drag and drop **Level 2** and **Level 3** into this hierarchy.

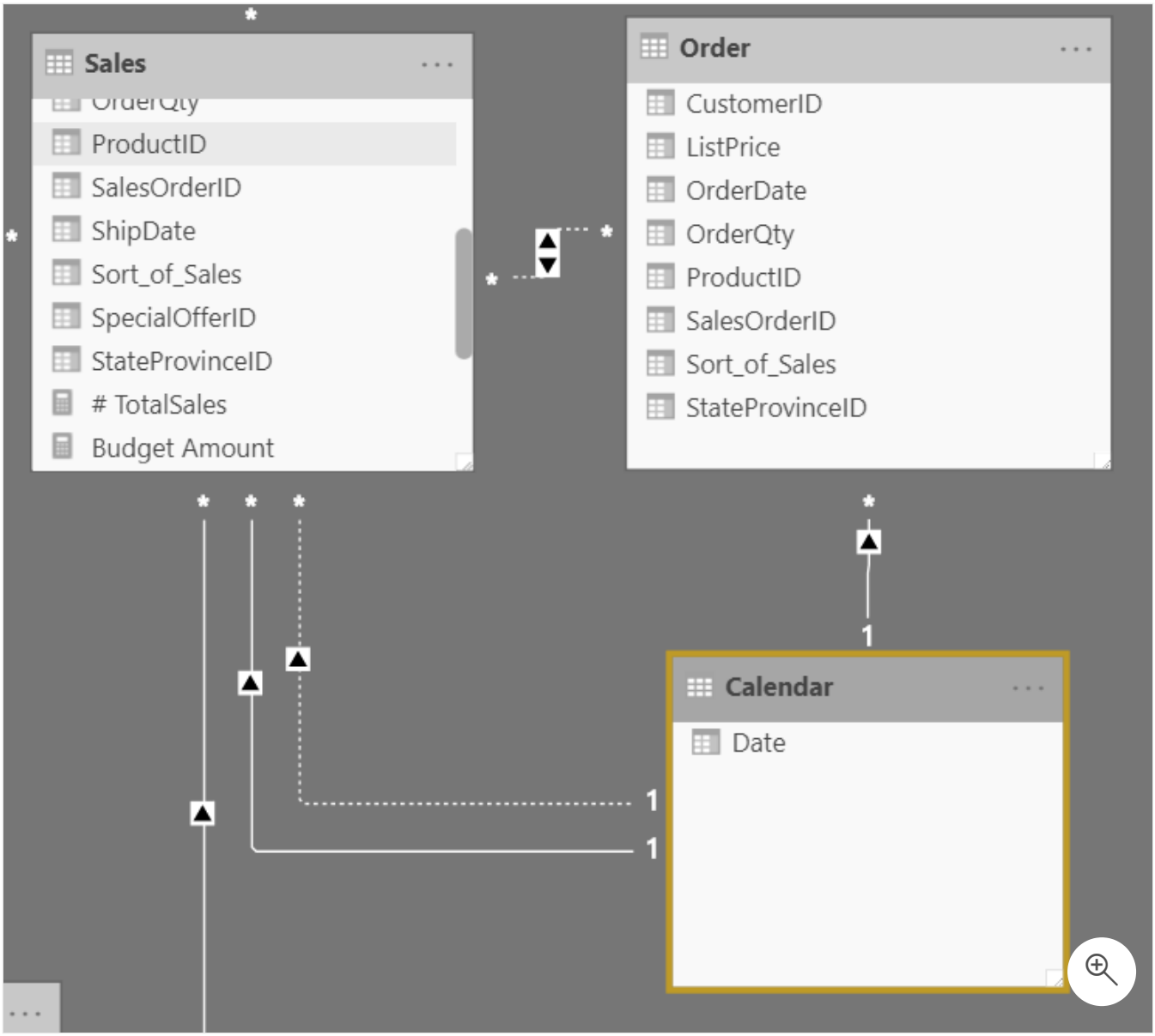


You have now successfully flattened a hierarchy so that you can view individual levels.

Previously, you've considered dimensions that have only one relationship with a fact table. However, situations do occur where your dimension table will have multiple relationships with a fact table.

Role-playing dimensions

Role-playing dimensions have multiple valid relationships with fact tables, meaning that the same dimension can be used to filter multiple columns or tables of data. As a result, you can filter data differently depending on what information you need to retrieve. This topic is complex, so it is only introduced in this section. Working with role-playing dimensions requires complex DAX functions that will be discussed in later sections.



The preceding visual shows the Calendar, Sales, and Order tables. Calendar is the dimension table, while Sales and Order are fact tables. The dimension table has two relationships: one with Sales and one with Order. This example is of a role-playing dimension because the Calendar table can be used to group data in both Sales and Order. If you wanted to build a visual in which the Calendar table references the Order and the Sales tables, the Calendar table would act as a role-playing dimension.

Next unit: Define data granularity

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

Define data granularity

4 minutes

Data granularity is the detail that is represented within your data, meaning that the more granularity your data has, the greater the level of detail within your data.

Data granularity is an important topic for all data analysts, regardless of the Power BI tools that you're using. Defining the correct data granularity can have a big impact on the performance and usability of your Power BI reports and visuals.

Data granularity defined

Consider a scenario where your company manages 1,000 refrigerated semi-trucks. Every few minutes, each truck uses a Microsoft Azure IoT application to record its current temperature. This temperature is important to your organization because, if the refrigeration were to malfunction, it could spoil the entire load, costing thousands of dollars. With so many trucks and so many sensors, extensive data is generated every day. Your report users don't want to sift through numerous records to find the ones that they are particularly interested in.

How can you change the granularity of the data to make the dataset more usable?

In this scenario, you might want to import the data by using a daily average for each truck. That approach would reduce the records in the database to one record for each truck for each day. If you decide that the approach was acceptable enough for tracking costs and errors, then you could use that data granularity. Alternatively, you could select the last recorded temperature, or you could only import records that are above or below a normal range of temperatures. Any of these methods will reduce the total records that you import, while still bringing in data that is comprehensive and valuable.

For different scenarios, you could settle on data granularity that is defined weekly, monthly, or quarterly. Generally, the fewer the records that you are working with, the faster your reports and visuals will function. This approach translates to a faster refresh rate for the entire dataset, which might mean that you can refresh more frequently.

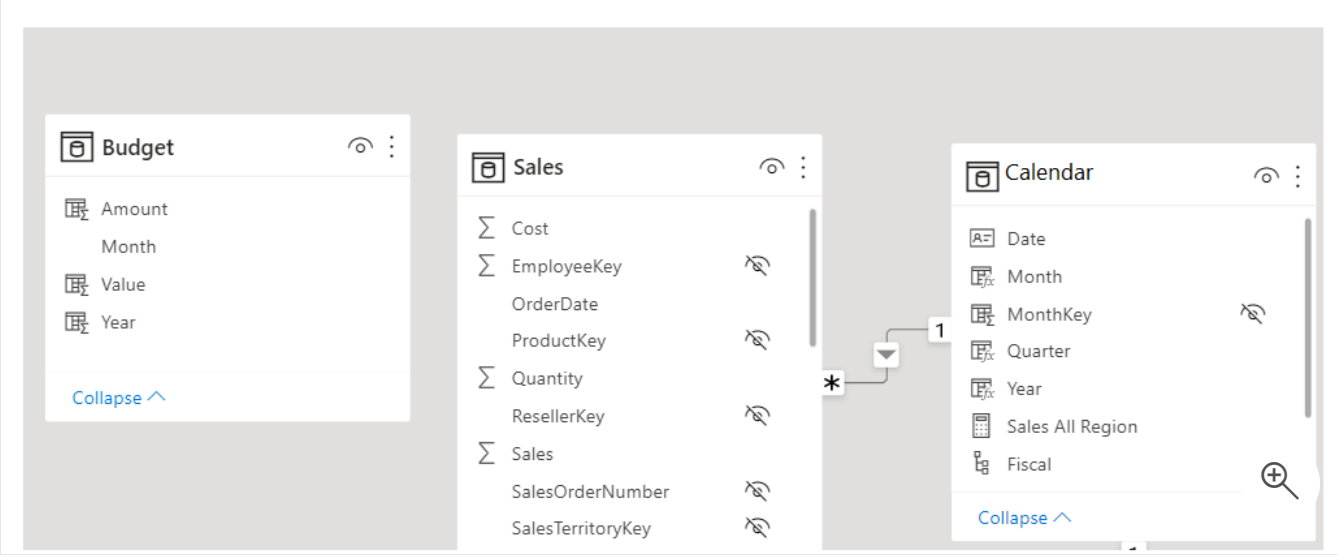
However, that approach has a drawback. If your users want to drill into every single transaction, summarizing the granularity will prevent them from doing that, which can have a negative impact on the user experience. It is important to negotiate the level of data granularity with report users so they understand the implications of these choices.

Change data granularity to build a relationship between two tables

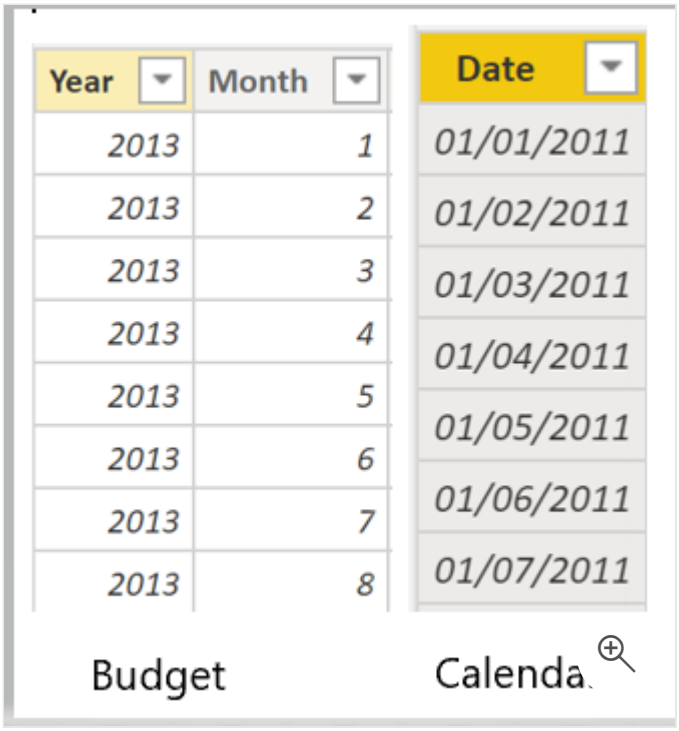
Data granularity can also have an impact when you're building relationships between tables in Power BI.

For example, consider that you're building reports for the Sales team at Tailwind Traders. You've been asked to build a matrix of total sales and budget over time by using the Calendar, Sales, and Budget tables. You notice that the lowest level of time-based detail that the Sales table goes into is by day, for instance 5/1/2020, 6/7/2020, and 6/18/2020. The Budget table only goes to the monthly level, for instance, the budget data is 5/2020 and 6/2020. These tables have different granularities that need to be reconciled before you can build a relationship between tables.

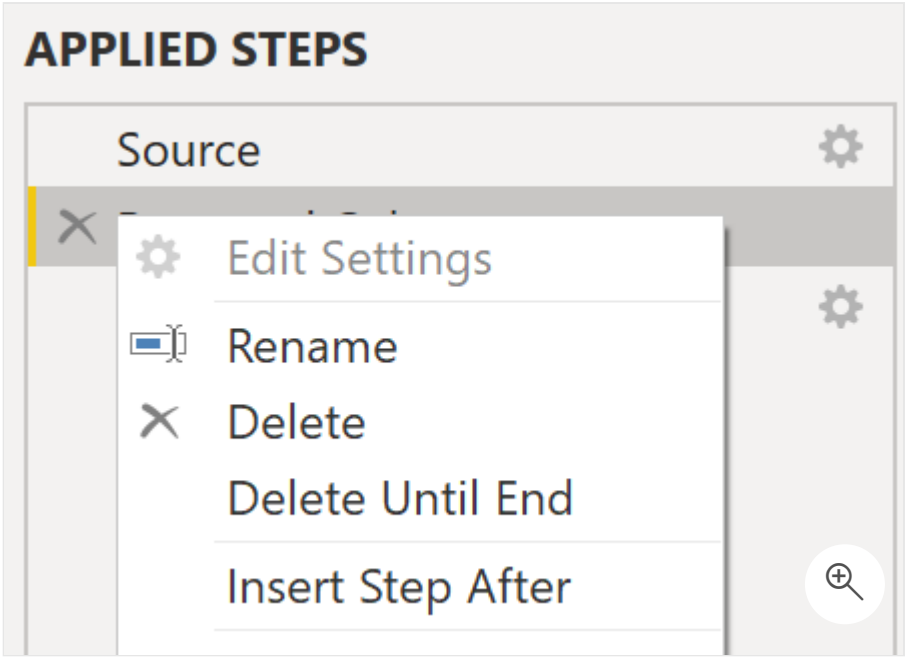
The following figure shows your current data model.



As shown in the preceding figure, a relationship between Budget and Calendar is missing. Therefore, you need to create this relationship before you can build your visual. Notice that if you transform the **Year** and **Month** columns in the Calendar table into a new column, and do the same transformation in the Budget table, you can match the format of the **Date** column in the Calendar table. Then, you can establish a relationship between the two columns. To complete this task, you'll concatenate the **Year** and **Month** columns and then change the format.



Select **Transform Data** on the ribbon. On **Applied Steps**, on the right pane, right-click the last step and then select **Insert Step After**.



Under **Add Column** on the Home ribbon, select **Custom Column**. Enter the following equation, which will concatenate the **Year** and **Month** columns, and then add a dash in between the column names.

M

```
Column = Table.AddColumn(#"Renamed Columns", "Custom", each [Year] & "-" &
[Month])
```

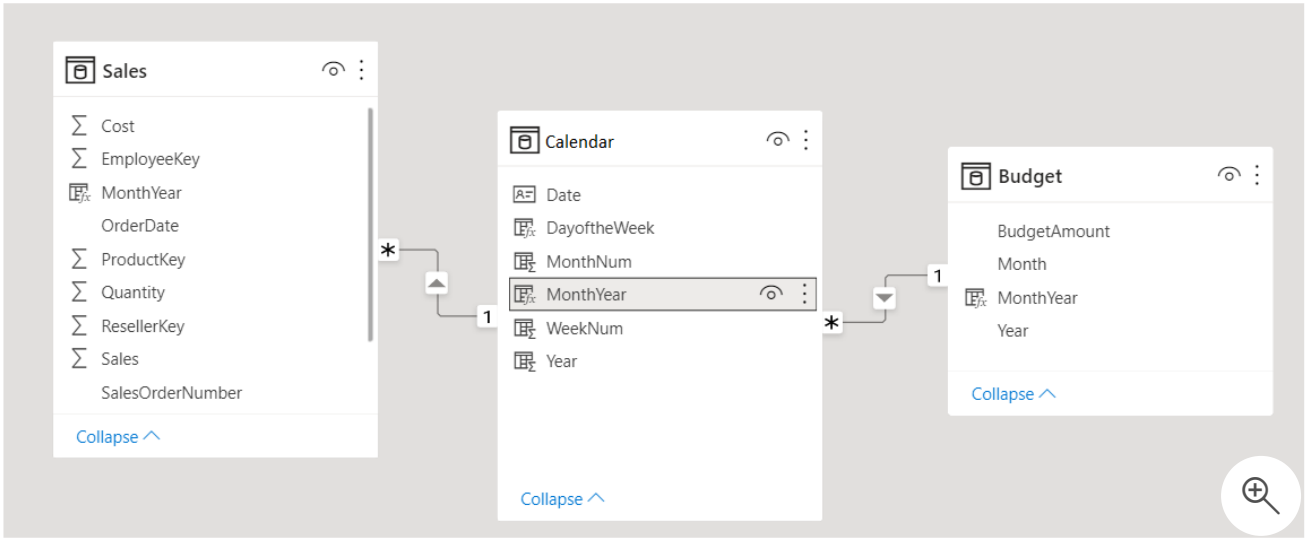
Change the data type to **Date** and then rename the column. Your Budget table should resemble the following figure.

Year	Month	BudgetAmount	MonthYear
2011	1	9493	2011-1
2011	2	2490	2011-2
2011	3	16585	2011-3
2011	4	5575	2011-4
2011	5	11611	2011-5
2011	6	14017	2011-6
2011	7	11919	2011-7
2011	8	3832	2011-8
2011	9	7094	2011-9
2011	10	6216	2011-10
2011	11	7769	2011-11
2011	12	12552	2011-12
2012	1	12781	2012-1
2012	2	7889	2012-2
2012	3	10329	2012-3
2012	4	5350	2012-4
2012	5	19216	2012-5
2012	6	5849	2012-6
2012	7	19269	2012-7
2012	8	3080	2012-8
2012	9	2381	2012-9

Now, you can create a relationship between the Budget and the Calendar tables.

Create a relationship between tables

Power BI automatically detects relationships, but you can also go to **Manage Relationships** > **New** and create the relationship on the **Date** column. The relationship should resemble the following figure.



By completing this task, you've ensured that the granularity is the same between your different tables. Now, you need to create DAX measures to calculate **Total Sales** and **BudgetAmount**. Go to the **Data** pane on Power BI Desktop, select **New Measure**, and then create two measures with the following equations:

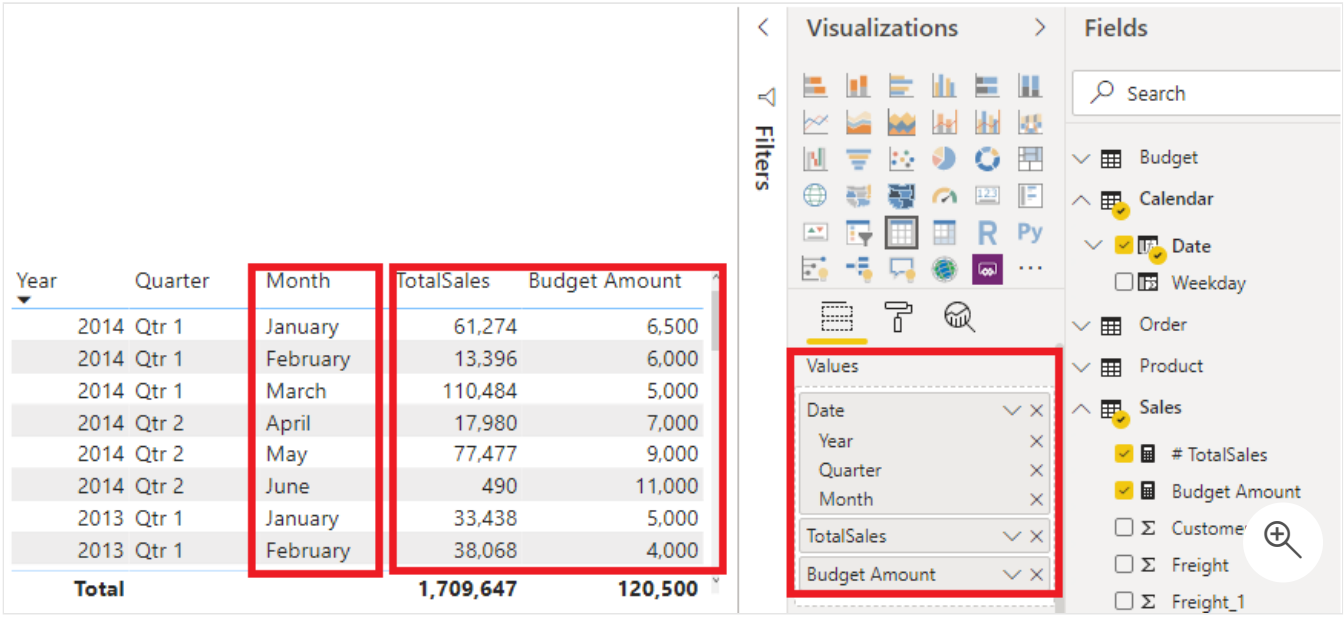
DAX

```
TotalSales = SUM(Sales[Total Sales])
```

DAX

```
BudgetAmount = SUM (Budget[BudgetAmount])
```

Select the table visual on the **Visualization** pane, and then enter these measures and the **Date** into the **Values** field. You've now accomplished the goal of building a matrix of the total sales and budgets over time.



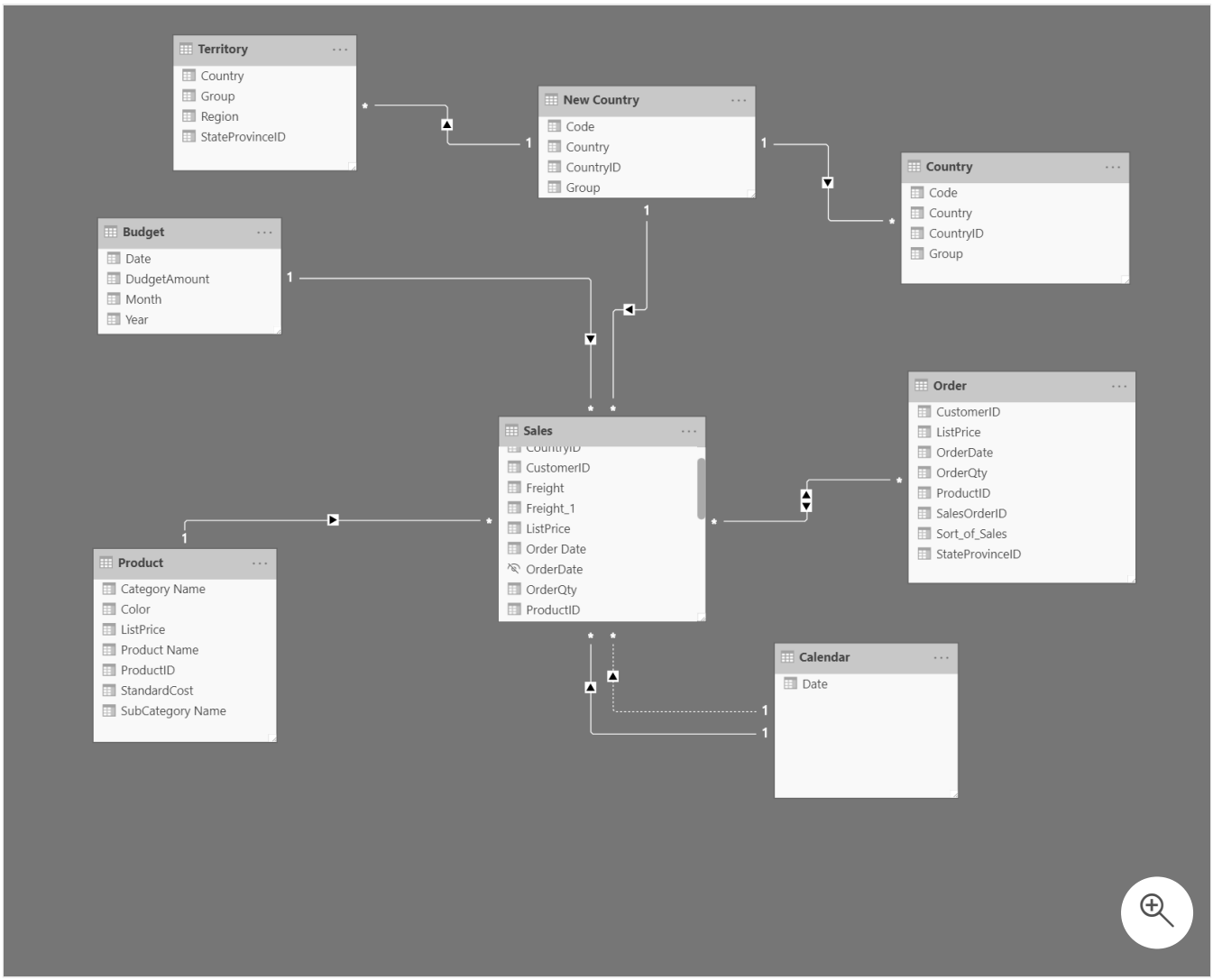
100 XP

Work with relationships and cardinality

5 minutes

Unlike other database management systems, Power BI has the concept of *directionality* to a relationship. This directionality plays an important role in filtering data between multiple tables. When you load data, Power BI automatically looks for relationships that exist within the data by matching column names. You can also use **Manage Relationships** to edit these options manually.

For example, you've retrieved many tables from the Sales database, and the following image is an example of your data model. Power BI has autodetected several relationships, but you can't discern what they mean. You want to make sure that the relationships accurately reflect those that exist in your data.



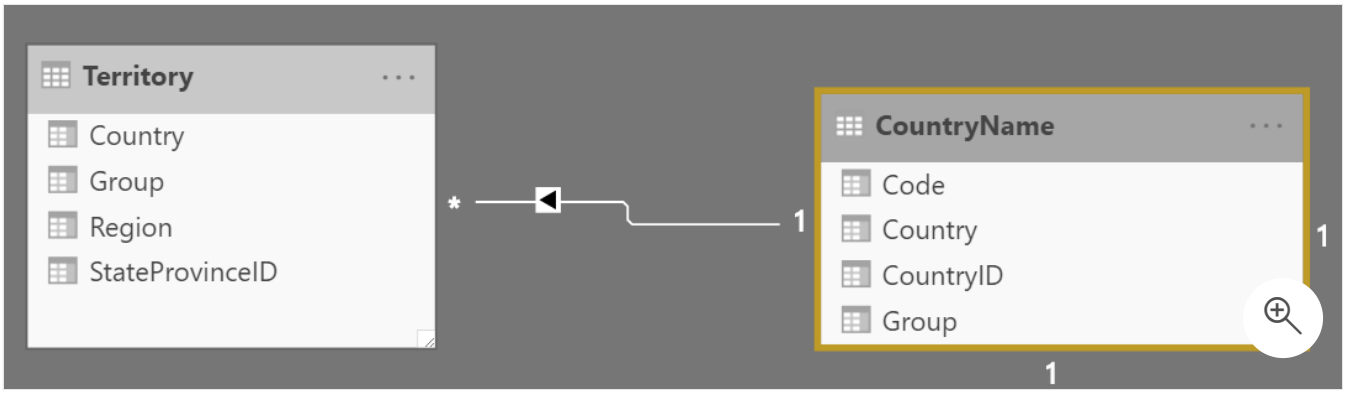
Relationships

The following are different types of relationships that you'll find in Power BI.

Many-to-one (*:1) or one-to-many (1: *) relationship

- Describes a relationship in which you have many instances of a value in one column that are related to only one unique corresponding instance in another column.
- Describes the directionality between fact and dimension tables.
- Is the most common type of directionality and is the Power BI default when you are automatically creating relationships.

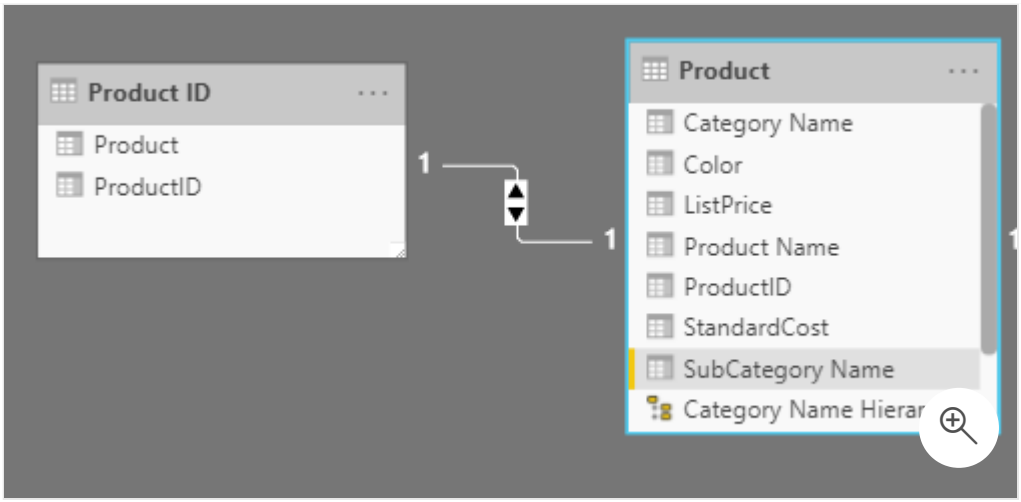
An example of a one-to-many relationship would be between the CountryName and Territory tables, where you can have many territories that are associated with one unique country.



One-to-one (1:1) relationship:

- Describes a relationship in which only one instance of a value is common between two tables.
- Requires unique values in both tables.
- Is not recommended because this relationship stores redundant information and suggests that the model is not designed correctly. It is better practice to combine the tables.

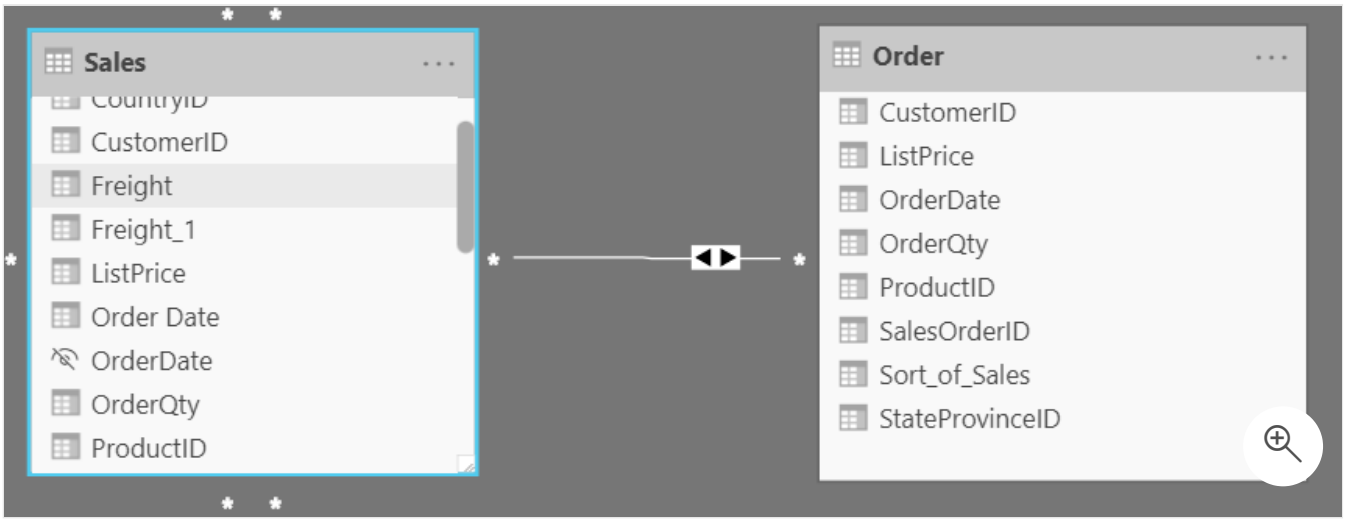
An example of a one-to-one relationship would be if you had products and product IDs in two different tables. Creating a one-to-one relationship is redundant and these two tables should be combined.



Many-to-many (.) relationship:

- Describes a relationship where many values are in common between two tables.
- Does not require unique values in either table in a relationship.
- Is not recommended; a lack of unique values introduces ambiguity and your users might not know which column of values is referring to what.

For instance, the following figure shows a many-to-many relationship between the Sales and Order tables on the **OrderDate** column because multiple sales can have multiple orders associated with them. Ambiguity is introduced because both tables can have the same order date.



Cross-filter direction

Data can be filtered on one or both sides of a relationship.

With a **single cross-filter direction**:

- Only one table in a relationship can be used to filter the data. For instance, Table 1 can be filtered by Table 2, but Table 2 cannot be filtered by Table 1.

  > **Tip**
  >
  > Follow the direction of the arrow on the relationship between your tables to know which direction the filter will flow. You typically want these arrows to point to your fact table.

- For a one-to-many or many-to-one relationship, the cross-filter direction will be from the "one" side, meaning that the filtering will occur in the table that has many values.

With **both cross-filter directions** or **bi-directional cross-filtering**:

- One table in a relationship can be used to filter the other. For instance, a dimension table can be filtered through the fact table, and the fact tables can be filtered through the dimension table.
- You might have lower performance when using bi-directional cross-filtering with many-to-many relationships.

A word of caution regarding bi-directional cross-filtering: You should not enable bi-directional cross-filtering relationships unless you fully understand the ramifications of doing so. Enabling it can lead to ambiguity, over-sampling, unexpected results, and potential performance degradation.

Cardinality and cross-filter direction

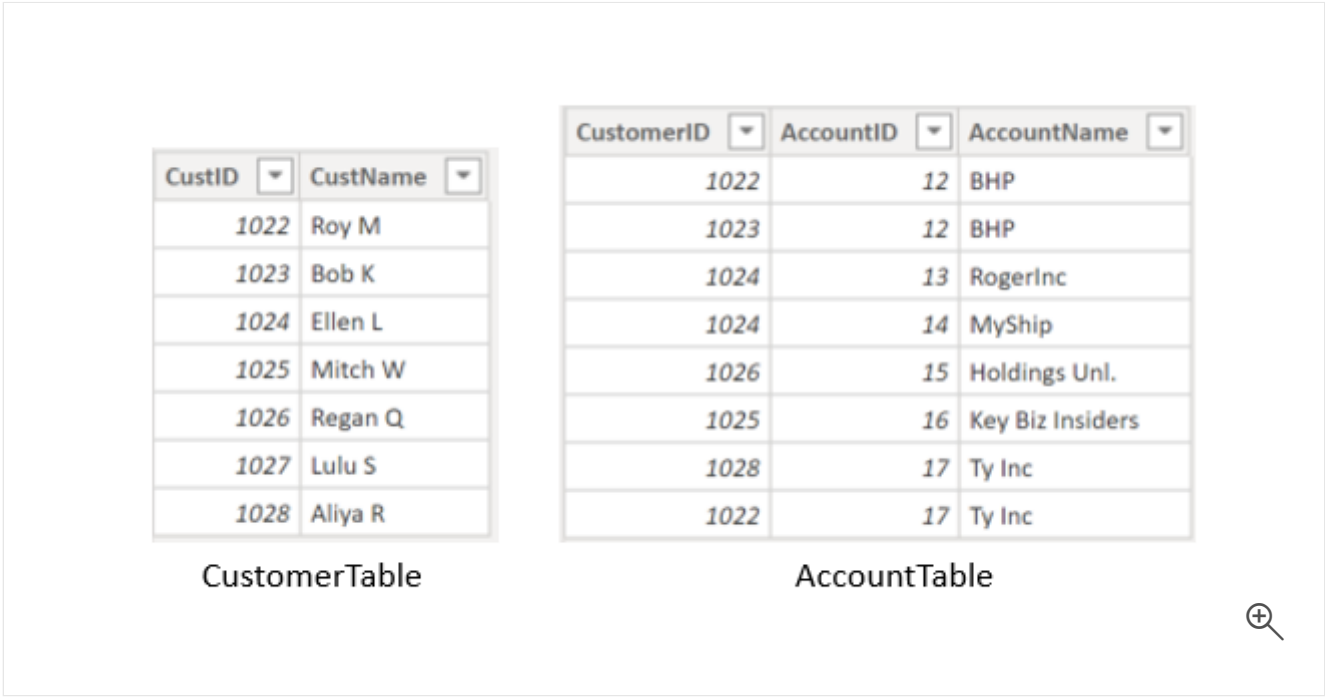
For one-to-one relationships, the only option that is available is bi-directional cross-filtering. Data can be filtered on either side of this relationship and result in one distinct, unambiguous value. For instance, you can filter on one Product ID and be returned a single Product, and you can filter on a Product and be returned a single Product ID.

For many-to-many relationships, you can choose to filter in a single direction or in both directions by using bi-directional cross-filtering. The ambiguity that is associated with bi-directional cross-filtering is amplified in a many-to-many relationship because multiple paths will exist between different tables. If you create a measure, calculation, or filter, unintended consequences can occur where your data is being filtered and, depending on which relationship that the Power BI engine chooses when applying the filter, the final result might be different. This situation is also true for bi-directional relationships and why you should be cautious when using them.

For this reason, many-to-many relationships and/or bi-directional relationships are complicated. Unless you are certain what your data looks like when aggregated, these types of open-ended relationships with multiple filtering directions can introduce multiple paths through the data.

Create many-to-many relationships

Consider the scenario where you are tasked with building a visual that examines budgets for customers and accounts. You can have multiple customers on the same account and multiple accounts with the same customer, so you know that you need to create a many-to-many relationship.



CustID	CustName
1022	Roy M
1023	Bob K
1024	Ellen L
1025	Mitch W
1026	Regan Q
1027	Lulu S
1028	Aliya R

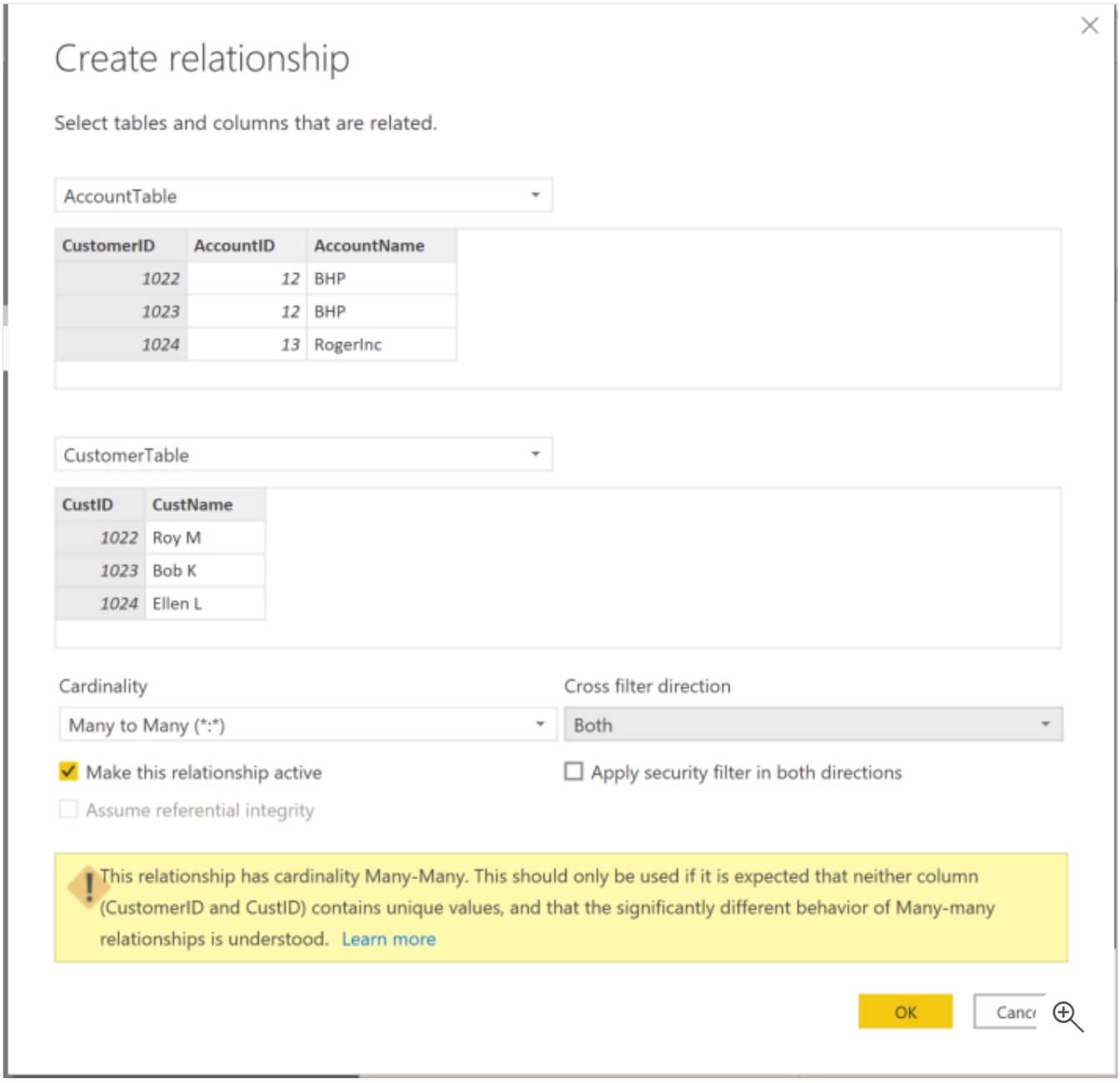
CustomerTable

CustomerID	AccountID	AccountName
1022	12	BHP
1023	12	BHP
1024	13	RogerInc
1024	14	MyShip
1026	15	Holdings Unl.
1025	16	Key Biz Insiders
1028	17	Ty Inc
1022	17	Ty Inc

AccountTable

To create this relationship, go to **Manage Relationships** > **New**. In the resulting window, create a relationship between the **Customer ID** column in CustomerTable and AccountTable. The relationship is set to many-to-many, and the filter type is in both directions. Immediately, you will be warned that you should only use this type of relationship if it is expected that neither column will have unique values because you might get unexpected values. Because you want to filter in both directions, choose **bi-directional cross-filtering**.

Select **OK**. You have now successfully created a many-to-many relationship.



For more information, see Many-to-many relationships in Power BI.

Next unit: Resolve modeling challenges

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Resolve modeling challenges

2 minutes

Modeling data is about establishing and maintaining relationships so that you can effectively visualize the data in the form that your business requires. When you are creating these relationships, a common pitfall that you might encounter are circular relationships.

For example, you are developing reports for the Sales team and are examining the relationships between tables. In a poorly designed data model, Table 1 has a many-to-one relationship with a column in Table 2, but Table 2 has a one-to-many relationship with Table 3 that has its own relationship with Table 1. This web of relationships is difficult to manage and becomes a daunting task to build visuals because it is no longer clear what relationships exist. Therefore, it is important that you are able to identify circular relationships so that your data is usable.

Relationship dependencies

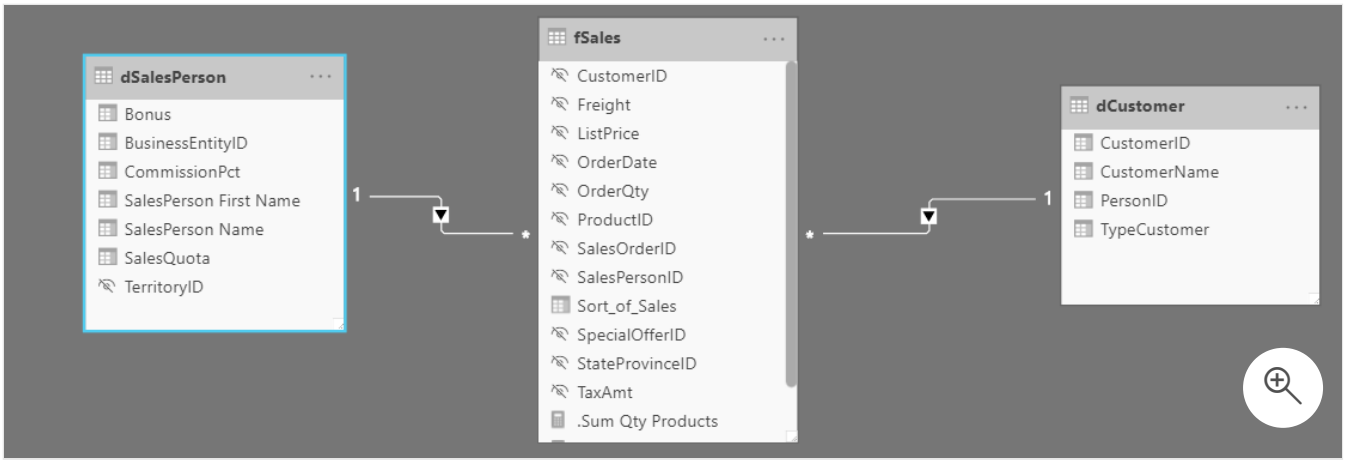
To understand circular relationships, you first need to understand dependencies.

For example, consider that you have the following calculated column **Total** in the Sales table.

Sales['TotalCost'] = Sales['Quantity'] * Sales['Price']

TotalCost depends on **Quantity** and **Price**, so if a change occurs in either quantity or price, a change will occur in **TotalCost** as well. This example outlines a dependency of a column on other columns, but you can also have dependencies between measures, tables, and relationships.

Consider the following relationships between **dSalesPerson**, **fSales**, and **dCustomer**. A change in **dCustomer** will result in a change in **fSales**, which results in changes in **dSalesPerson.** These types of dependencies can exist within relationships.



Next unit: Exercise - Model data in Power BI Desktop

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

200 XP

Check your knowledge

3 minutes

Answer the following questions to see what you've learned.

1. What does data granularity mean? *

- Data granularity is the filter direction that is associated with the relationship between two columns.
- Data granularity is the level of detail that is represented in the data.

  ✔ **Correct. Data granularity indicates the level the detail within the data, so the higher the granularity, the more detail that is represented in the data.**
- Data granularity is a type of schema design.

2. What is the difference between a fact table and a dimension table? *

- Fact tables contain observational data while dimension tables contain information about specific entities within the data.

  ✔ **Correct. Fact tables contain observational data such as sales orders, employees, shipping dates, and so on, while dimension tables contain information about specific entities such as product IDs and dates.**
- Fact tables contain information about specific entities while dimension tables contain information about observational data.
- There is no difference.

3. Choose the best answer to explain relationship cardinality? *

- Cardinality is the measure of unique values in a table.

  ✔ **Correct. An example of high cardinality would be a Sales table as it has a high number of unique values.**

- ◯ Cardinality is the granularity of the data.
- ◯ Cardinality is how long it takes for the data to load.

Next unit: Summary

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

Practice Assessment for Exam PL-300: Microsoft Power BI Data Analyst

Question 29 of 50

You have designed a star schema to simplify your data.

You need to understand the relationship between the tables in the star schema.

What is the relationship between the fact table and dimension tables?

- ○ many-to-many
- ○ many-to-one

✓This answer is correct.

- ○ one-to-many

This answer is incorrect.

- ○ one-to-one

There is a many-to-one relationship between a fact table and dimension table. For each row in a dimension table there may be multiple matching rows in the fact table.

Introduction - Training | Microsoft Learn

Next >

Check Your Answer

Practice Assessment for Exam PL-300: Microsoft Power BI Data Analyst

Question 30 of 50

You have the following Power Query M formula that generates a range of dates.

```
= List.Dates(#date(2020,05,31), 365, #duration(1,0,0,0))
```

What is the resulting date range?

- ◯ a list of days ending on May 31, 2020 and starting 365 days earlier
- ◉ a list of days starting on May 31, 2020 and ending 365 days later

✓This answer is correct.

- ◯ a list of months ending in May 2020 and starting 12 months earlier
- ◯ a list of months starting in May 2020 and ending 12 months later

The `#date` element designates the starting date, `365` designates the duration counter, and `#duration(1,0,0,0)` designates the duration interval in days, so this formula results in a listing of days starting on May 31, 2020 and ending 365 days later.

Create a date table - Training | Microsoft Learn

Next >

Check Your Answer

Practice Assessment for Exam PL-300: Microsoft Power BI Data Analyst

Question 31 of 50

You need to prevent hidden date tables from being auto generated by Power BI Desktop for every date or datetime data type column in a dataset.

What two tasks should you perform? Each correct answer presents a complete solution

- [x] Enable Mark as date table for the Calendar table.

✓This answer is correct.

- [x] From the Current File options in Power BI Desktop, disable Auto Date/Time

✓This answer is correct.

- [ ] From the Global options in Power BI Desktop, disable Auto Date/Time for new files.
- [ ] Set the Data Category to None for all Date and DateTime fields.

Disabling Auto Date/Time for new files from the Current File options will disable all Auto Date/Time tables in this dataset. Enabling Mark as date table for the Calendar table will also disable the auto datetime tables in the dataset. Disabling the Global option Auto Date/Time for new files means that new files will no longer have Auto Date/Time enabled, but the file containing the current dataset will still have it enabled until it is disabled. Changing the data category will not impact the auto date table feature.

Create a date table - Training | Microsoft Learn

Auto date/time in Power BI Desktop - Power BI | Microsoft Learn

Auto date/time guidance in Power BI Desktop - Power BI | Microsoft Learn

Next >

Check Your Answer

Practice Assessment for Exam PL-300: Microsoft Power BI Data Analyst

Question 32 of 50

You have a Power BI model.

You need to assign items to a display folder.

Which three items can be assigned to a display folder? Each correct answer presents part of the solution.

- [x] Calculated column

✓This answer is correct.

- [x] Column

✓This answer is correct.

- [x] Measure

✓This answer is correct.

- [ ] Report
- [ ] Table

This answer is incorrect.

Tables and reports cannot be assigned a display folder. Columns, calculated columns, and measures can be assigned a display folder.
Lab - Model data in Power BI Desktop, Part 1 - Training | Microsoft Learn

Next > Check Your Answer

Practice Assessment for Exam PL-300: Microsoft Power BI Data Analyst

Question 33 of 50

You have a Power BI Desktop dataset that includes a table named Employees containing a row for each employee with the following columns:

- Employee ID
- Employee Name
- Manager ID
- Manager Name

You need to flatten the parent-child hierarchy in the Employees table by adding an extra column that will contain a listing of employee IDs for all direct and indirect managers of each employee.

Which two Data Analysis Expression (DAX) functions should you use? Each correct answer presents part of the solution.

- [ ] `PATHCROSSJOIN`

This answer is incorrect.

- [ ] `EXCEPT PATHITEM`
- [ ] `EXCEPTPATH`

✓This answer is correct.

- [ ] `PATHITEM CROSSJOIN`

✓This answer is correct.

- [ ] `RELATED`

This answer is incorrect.

The `PATH` function returns a string with identifiers of all the parents of the current identifier, which is used for flattening. The `PATHITEM` function returns the item at the specified position of a string, which is also used for flattening. The `EXCEPT` function returns rows from one table which do not appear in another table, which

would require another table. The `CROSSJOIN` function returns a Cartesian product of all rows from all tables that the function references. The `RELATED` function returns a related value from another table, which would require another table.
Design a data model in Power BI - Training | Microsoft Learn

Next >

Check Your Answer